Methodology of Multi-FPGA Prototyping Platform Generation

Qingshan Tang

UPMC SORBONNE UNIVERSITÉS

THÈSE DE DOCTORAT DE l'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Qingshan TANG

Pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse :

MÉTHODOLOGIE DE GÉNÉRATION DE PLATEFORME DE PROTOTYPAGE À BASE DE MULTI-FPGA

Soutenue le 13 Janvier 2015

Devant le jury composé de :

M.	Laurent FESQUET	MCF	HDR Lab TIMA	Rapporteur
M.	Adel BAGANNE	MCF	HDR Lab-STICC	Rapporteur
M.	Omar HAMMAMI	MCF	HDR ENSTA ParisTech	Examinateur
M.	Gérard SOU	MCF	HDR UPMC	Examinateur
Mme.	Helena KRUPNOVA	PhD	Synopsys	Examinateur
M.	Habib MEHREZ	Professeur	UPMC	Directeur de thèse
M.	Matthieu TUNA	PhD	Flexras Technologies	Co-Directeur de thèse

Ph.D THESIS OF PIERRE-AND-MARIE-CURIE UNIVERSITY

Department:

Computer Science

École Doctorale Informatique, Télécommunications et Électronique (Paris)

Presented by

Qingshan TANG

Thesis submitted to obtain the degree of

Ph.D OF PIERRE-AND-MARIE-CURIE UNIVERSITY

METHODOLOGY OF MULTI-FPGA BASED PROTOTYPING PLATFORM GENERATION

Defense date: 13 January 2015

Committee in charge:

M.	Laurent FESQUET	MCF	HDR Lab TIMA	Rapporteur
M.	Adel BAGANNE	MCF	HDR Lab-STICC	Rapporteur
M.	Omar HAMMAMI	MCF	HDR ENSTA ParisTech	Examinateur
M.	Gérard SOU	MCF	HDR UPMC	Examinateur
Mme.	Helena KRUPNOVA	PhD	Synopsys	Examinateur
M.	Habib MEHREZ	Professeur	UPMC	Directeur de thèse
M.	Matthieu TUNA	PhD	Flexras Technologies	Co-Directeur de thèse

Résumé

Face à la difficulté grandissante de l'intégration matériel/logiciel, le prototypage à base de cartes multi-FPGA devient obligatoire dans l'arsenal des techniques de vérification pré-silicium. Les plateformes de prototypage multi-FPGA peuvent être classées en trois catégories: sur étagère, sur mesure et câblées. La plateforme sur étagère se compose d'une carte multi-FPGA prêt à l'emploi et générique. Toutes les connexions inter-FPGA sont fixées et réalisées à l'aide de pistes sur le PCB. La plateforme sur mesure se compose d'une carte multi-FPGA conçue spécifiquement pour un design donné. Les connexions inter-FPGA sont également réalisées à l'aide de pistes PCB. Les plateformes câblées, qui se composent de plusieurs cartes mono-FPGA prêt à l'emploi, sont connectées grâce à des câbles peuvent être connectées/déconnectées à volonté. Elles peuvent être considérées comme "semi-sur étagère" par le fait qu'elles sont constituées de plusieurs cartes prêt à l'emploi, et "semi-sur mesure" par le fait que les connexions entre FPGAs sont définies par l'utilisateur et adaptées au design testé.

Nous avons dégagé dans ce manuscrit trois problèmes majeurs en matière de prototypage à base de cartes multi-FPGA: (1). L'évolution des FPGA tend à faire des entrées/sorties (E/S) une ressource rare, aggravant le problème de bande passante inter-FPGA génération après génération. En effet, les plateformes multi-FPGA souffrent des délais de communication importants inter-FPGA par rapport aux délais intra-FPGA. Par conséquent, il devient de plus en plus difficile de prototyper un design SoC/ASIC à des performances intéressantes. (2). 70% des plateformes de prototypage multi-FPGA sont des plateformes sur mesure en raison des besoins de performance et de coût. Néanmoins, la définition d'une plateforme sur mesure est aujourd'hui un processus majoritairement manuel et chronophage. Ainsi, l'exploration de cartes avec des types de FPGA différents, qui permettrait aux ingénieurs de concevoir une plateforme de prototypage optimale, ne peut pas être faite. Comme le rapport entre la capacité logique et le nombre d'E/S des FPGA est en augmentation à un rythme quasi-exponentiel, il devient de plus en plus difficile de concevoir une plateforme sur mesure performante. (3). La plateforme câblée bénéficie de la disponibilité et de la personnalisation. Les performances d'une plateforme câblée dépendent de la distribution des câbles et du placement des interfaces externes. Néanmoins, il n'existe pas d'outil permettant d'obtenir automatiquement une solution pour la distribution des câbles. Une distribution de câbles permettant d'obtenir des fréquences de fonctionnement élevées devient de plus en plus difficile à atteindre en raison de la limitation des E/S. Par rapport aux plateformes sur étagère, la valeur ajoutée, en terme de performance, des plateformes câblées et sur mesure peuvent être fortement dégradée par une définition des interconnexions entre FPGA inefficace.

Les contributions de ce manuscrit sont: (1). Un nouvel algorithme de routage exploitant les pistes multi-points connectant plus de deux FPGA, permettant ainsi d'augmenter les performances. (2). Un flot de conception automatique permettant de créer une plateforme sur mesure, augmentant ainsi la productivité et permettant l'exploration de cartes. (3). Une architecture de plateforme câblée est proposée ainsi qu'un algorithme permettant automatiquement de trouver une solution pour la distribution des câbles. (4). Finalement une comparaison entre ces différentes plateformes est réalisée d'un point de vue quantitatif et qualitatif.

Mots-clés : Circuit Intégré, Vérification, Multi-FPGA, Prototypage, Sur Mesure, Câblée

Abstract

Multi-FPGA based prototyping is no longer optional for hardware/software integration. We can classify multi-FPGA prototyping platforms in three categories: off-the-shelf, custom and cabling. The off-the-shelf platform consists of a ready-made generic multi-FPGA board, where all the inter-FPGA connections are fixed and realized using PCB traces. The custom platform consists of a build-your-own multi-FPGA board tailored for a specific design, where all the inter-FPGA connections are realized using PCB traces as well. The cabling platform consists of multiple ready-made FPGA boards connected by cables and connectors. The cabling platform is semi off-the-shelf due to that it consists of multiple ready-made boards, and semi custom due to that its connections inter FPGAs as well as connections to external interfaces are user-defined and tailored for a specific design.

There are three existing problems regarding to multi-FPGA based prototyping: (1). FPGA I/Os are becoming a scarce resource, worsening the inter-FPGA bandwidth generation after generation. Unfortunately, multi-FPGA platforms suffer from large timing delays in inter-FPGA communication compared to intra-FPGA net delays. Therefore, it becomes more and more difficult to prototype an SoC/ASIC design at proper performance. (2). 70% of multi-FPGA prototyping platforms are home-made custom platforms due to performance requirement, external interfaces, and cost. Nevertheless, crafting a home-made custom multi-FPGA platform is today a manual process, thus, time-consuming. The board exploration with different FPGA types, which helps the engineers to design an optimum prototyping platform, can not be done. As the ratio between the logic capacity and the number of FPGA I/Os is increasing at an exponential rate, it becomes more and more challenging to design a high-performance custom multi-FPGA platform. (3). The cabling platform benefits from the availability and the customization. The performance of the cabling platform depends on the distribution of the cables and the placement of the external interfaces. Nevertheless, there is no tool to automatically have a solution for the cable distribution and external interface placement. A high-performance cables distribution becomes more and more difficult to be achieved due to the pin limitation. Compared to the off-the-shelf platform, the added value, in terms of performance, of cabling or custom platforms can be heavily impaired by an inefficient board design.

The contributions of the manuscript are: (1). A new routing algorithm is proposed to spare FPGA I/Os by exploiting multi-point tracks that connect more than two FPGAs, thus increasing the performance. (2). An automatic design flow for creating a custom platform is proposed, thus increasing the productivity, enabling the board exploration, and optimizing cost and performance. (3). The cabling platform is proposed where one board is composed of one FPGA and several connectors. The connections between FPGAs as well as the connections to external interfaces can be added or removed by only connecting or disconnecting the cables (resp. daughter boards) with or from the connectors. Then, an algorithm is proposed to automatically find a solution for the cable distribution. (4). Thanks to the developed automatic tools, the achieved performances for a set of designs mapped on the three different categories of multi-FPGA platforms are compared. The performance gains between these platforms are quantified.

Keywords: Integrated Circuit, Verification, Multi-FPGA, Prototyping, Custom, Cabling

à mes parents

Remerciements

Je souhaite remercier en premier lieu mon directeur de thèse, M. Habib MEHREZ, professeur et responsable de l'équipe CIAN du département SoC du LIP6 de UPMC, pour m'avoir accueilli au sein de son équipe.

J'adresse mes remerciements les plus chaleureux à M. Matthieu TUNA, PhD ingénieur de la société Flexras Technologies, pour tous les précieux conseils qu'il m'a donnés, pour la confiance qu'il m'a témoigné et sans qui ce travail n'aurait pas vu le jour. Je lui sui également reconnaissant pour sa disponibilité, ses qualités pédagogiques et scientifiques. J'ai beaucoup appris à ces côtés et je lui adresse toute ma gratitude.

Je voudrais remercier les rapporteurs de cette thèse M. Laurent FESQUET, maître de conférence au laboratoire TIMA, et M. Adel BAGANNE, maître de conférence au laboratoire STICC, pour l'intérêt qu'ils ont porté à mon travail.

J'associe à ces remerciements M. Omar HAMMAMI, maître de conférence à ENSTA Paristech, M. Gérard SOU, maître de conférence à UPMC, et Mme. Helena KRUPNOVA, PhD chef de projet chez Synopsys, pour avoir accepté d'examiner mon travail.

Je tiens à remercier tous mes camarades de la société Flexras Technologies, M. Hayder MRABET, M. Matthieu TUNA, M. Zied MARRAKCHI, M. Christophe ALEXANDRE, M. Ramsis FARHAT, M. Fathi LAYOUNI, M. Christian MASSON, pour avoir eu la patience d'écouter et de répondre à toutes mes questions techniques, et pour nos longues conversations sur la vie.

J'exprime ma gratitude aussi à mes collègues du LIP6, en particulier à Mme. Mariem TURKI, M. Alp KILIC, et M. Vinod PANGRACIOUS, pour leur soutien et leur bonne humeur.

Enfin je remercie mes parents pour leur soutien au cours de ces longues années d'études et sans lesquels je n'en serai pas là aujourd'hui.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Thesis Motivation

The continuous improvement of integrated circuit technologies leads to develop more complex and higher performance circuits. As a consequence, more and more efforts are required for their verification.

1.1.1 Design Verification

According to International Business Strategies, Inc. (IBS [IBS, 2009]), the cost of verification is growing at an exponential rate and has already been the highest portion of the overall design cost (~ 70%) as shown in the Figure 1.1.

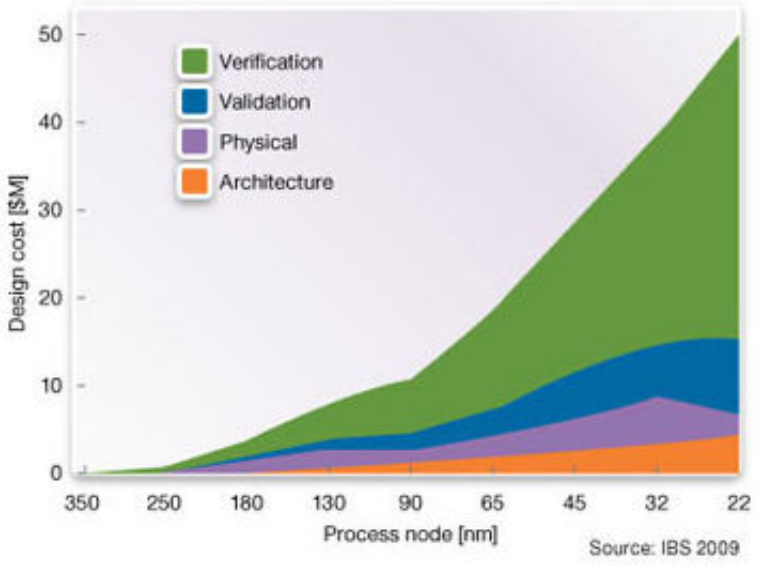


Figure 1.1: *Overall design cost*

1.1.2 Different Verification Techniques Pre-Silicon

There are four main verification techniques pre-silicon: simulation, emulation, virtual prototyping and FPGA-based prototyping. Each method plays its own role in the verification process as shown in Figure 1.2:

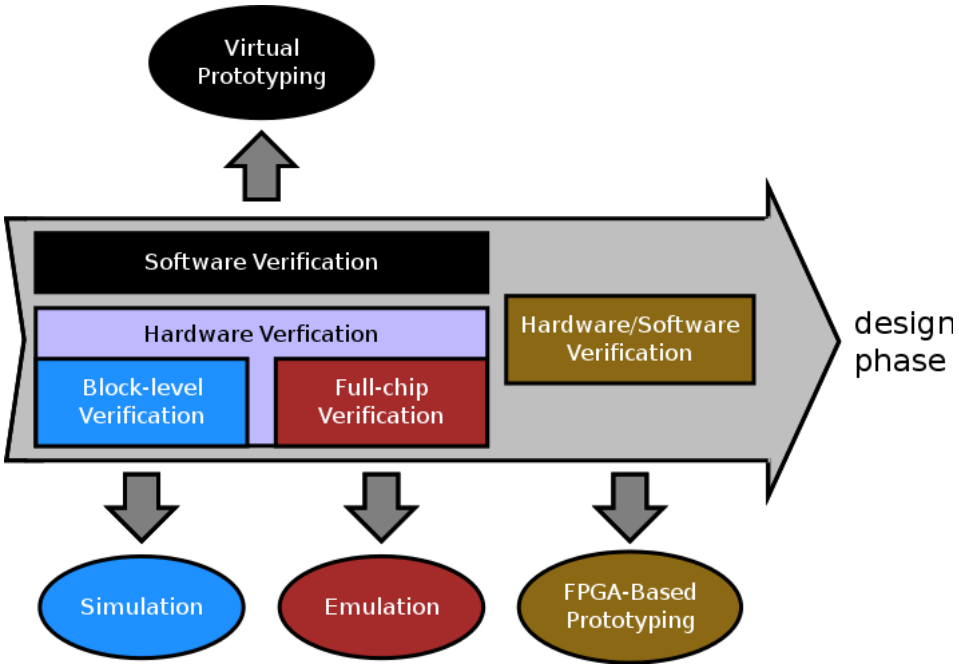


Figure 1.2: *Verification technology*

- Simulation: Simulators are being relegated to verifying design functionality at the block level of an SoC/ASIC (Block-level Verification). This technique offers full-visibility into the RTL, the set-up for implementation is instantaneous and the cost is about thousands of dollars. Nevertheless, the execution speed is very limited ($\sim 1KHz$). Simulators are software tools on a host PC. The three major existing software simulators are Cadence Incisive Enterprise Simulator [Incisive, 2014], Mentor Graphics ModelSim/SE [ModelSim, 2014], and Synopsys VCS [VCS, 2014].

- Emulation: Emulators are deployed to simulate an entire SoC/ASIC (Full-chip Verification) and thereby address the limitations of simulators. In the emulation, the RTL is still unstable (and therefore requires detailed debug analysis and frequent overnight recompiles). This technique provides high-visibility into the RTL and has much faster execution speed than the simulation ($\sim 1MHz$). Nevertheless, it is the most expensive (about millions of dollars) and takes several days to several weeks for set-up. Emulators are hardware platforms as shown in Figure 1.3. The three major existing hardware emulator platforms are Cadence Palladium emulator [Palladium, 2014], Mentor Graphics Veloce Emulation Systems [Veloce, 2014], and Synopsys EVE ZeBu Emulation [ZeBu, 2014].

- Virtual Prototyping: Virtual prototype is the earliest available Software Verification in the project, permitting co-development of hardware along with early software (i.e. If the virtual prototype shows that there is not enough processing bandwidth for concurrent applications, extra or different CPUs might be added). This technique represents fully functional software models of SoC/ASICs, boards, virtualized I/Os, external interfaces, all running on a host PC. But these software models are loosely-timed. Virtual prototyping offers good system visibility and control, which is especially useful for debug. Nevertheless, due to that it is not cycle-accurate, it can not ensure the functionality of the software in the hardware. As simulators, virtual prototypes are software tools on a host PC. The three major existing virtual prototype tools are Cadence Virtual System Platform [Virtual, 2014], Mentor Graphics Vista [Vista, 2014], and Synopsys Virtualizer [Virtualizer, 2014].

- FPGA-Based Prototyping: FPGA-based prototypes permit to run the software at almost real-time speed in a cycle-accurate and bit-accurate model of the SoC/ASIC (Hardware/-Software Verification). In the FPGA-based prototyping, the SoC/ASIC RTL is relatively stable. This technique is the one that offers the best execution speed ($\sim 10MHz$) but the least visibility compared to other hardware-related verification techniques. It costs about ten thousands of dollars and takes several weeks to several months for set-up. Different from the Simulation/Emulation, FPGA-Based prototyping enables "real world" testing. The prototyped SoC/ASICs are put into actual hardware and real external interfaces (such as DDR, PCI, Ethernet and etc.) are used. Implementing a design into a FPGA-based platform is a challenging process. As emulators, FPGA-based prototypes are hardware platforms as shown in Figure 1.3. Nevertheless, it is smaller in form factor, which makes it useful for software developers. One or several software developers can have one in each desktop. The major existing FPGA-based prototype providers are:

  - Cadence Protium Rapid Prototyping Platform [Protium, 2014] affords both the hardware platform and the implementation tool.
  - Synopsys supplies both the hardware platform HAPS [HAPS, 2014] and the implementation tool ProtoCompiler [ProtoCompiler, 2014].
  - S2C [S2C, 2014] sells both the hardware platform and the implementation tool.
  - Dini Group produces only the hardware platform [DINI, 2014].
  - HyperSilicon offers only the hardware platform [HyperSilicon, 2014].
  - Auspy accommodates only the implementation tool [Auspy, 2014].
  - Flexras Technologies provides only the implementation tool Wasga [Flexras, 2014].

Figure 1.3: *Emulator versus FPGA-based prototype in form factor*

1.1.3 FPGA-Based Prototyping

As the complexity of System on Chip (SoC) circuits and the quantity of software to be developed are increasing, the software developers can no longer wait for the chip to be fabricated for the integration of the hardware/software phase in order to meet the ever-shrinking time-to-market window [Huang et al., 2011]. The resulting trend is that FPGA-based prototyping is no longer optional [Amos et al., 2011].

1.1.3.1 Advantages of FPGA-based prototyping

High performance and timing-accuracy

Only FPGA-based prototyping provides both the speed and timing-accuracy necessary to properly test many aspects of the SoC/ASIC design (such as: real-time dataflow, hardware/software integration).

- Real-time dataflow: One of the reason that verifying an SoC is hard is due to that its state depends on many variables, including its previous state, the sequence of inputs and the wider system effects (and possible feedback) of the SoC outputs. Running the SoC design at almost real-time speed connected into the rest of the system (with "real world" interfaces) allows us to see the immediate effect of real-time conditions, inputs and system feedback when they change.

- Hardware/Software integration: Due to that software has already come to dominate SoC development effort, it is increasingly common that the software effort is on the critical path of the project schedule. FPGA-based prototyping offers a hardware model, the closest to the future silicon for testing hardware/software integration.

High mobility

Due to the small form factor of FPGA-based prototyping and its ability to work standalone, it has high mobility. It is often used for pre-production demonstration of new product capabilities at trade shows.

1.1.3.2 Disadvantages of FPGA-based prototyping

Low visibility

Even though there are many ways to instrument an FPGA in order to gain some visibility into design's functionality, it is still only a fraction of the information that is readily available in a simulator or an emulator. Therefore, we should always wait until the SoC/ASIC designs' RTL is fairly mature in simulation or emulation before passing it over to the FPGA-based prototyping.

Long set-up time

Due to that SoC/ASIC designs are often FPGA-hostile, the RTL modifications for FPGA-based prototyping need to be done as follows:

- The top-level pads of SoC/ASICs need to be adapted for the FPGA tool flow.
- The gated-clock and the complex generated clocks in SoC/ASICs need to be converted in FPGAs.
- Deal with some SoC design elements that are not available in FPGAs such as analog circuitry, BIST, SoC primitive and third party IP.
- Memories in SoC/ASICs need to be handled with FPGA memory resources (i.e. block RAM and distributed RAM) or external memory resources.

Therefore, the set-up time is long for FPGA-based prototyping.

Even though FPGA-based prototyping is not ideal, it provides a unique pre-silicon model of target silicon, allowing the software to be introduced to a cycle-accurate and high performance model of the hardware as early as possible.

1.1.4 Multi-FPGA Platform

Due to that the silicon area overhead of FPGA versus ASIC technology has been measured to be about 40x [Kuon and Rose, 2010], FPGA technology requires that an ASIC logic design should be partitioned across multiple FPGA devices to achieve the necessary logic capacity. The number of FPGAs depends on the size of the prototyped SoC/ASIC, ranging from a few [Krupnova, 2004] up to 60 FPGAs [Asaad et al., 2012].

According to their role and characteristics, we have classified the multi-FPGA prototyping platforms in three major categories:

- **Hardwired Off-the-Shelf Platform** [DINI, 2014] [GiDEL, 2014] [POLARIS, 2014] (abbreviation: off-the-shelf): consists of a ready-made generic multi-FPGA board, where all the inter-FPGA connections are fixed and realized using PCB traces. The connections to external interfaces are also fixed but can be realized using PCB traces or connectors.

- **Hardwired Custom Platform** [ReFLEX, 2014] (abbreviation: custom): consists of a build-your-own specific multi-FPGA board, where all the inter-FPGA connections are realized using PCB traces as well. Different from the off-the-shelf platform that has generic and balanced connections, the connections inter FPGAs as well as the connections to external interfaces of the custom platform are user-defined and tailored for a specific design.

- **Cabling Platform** [HAPS, 2014] [Prodesign, 2014]: which is a relatively new notion compared to other two platforms, consists of multiple ready-made FPGA boards connected by cables and connectors. The cabling platform, which is in between the off-the-shelf and the custom platform as shown in Figure 1.4, is semi off-the-shelf due to that it consists of multiple ready-made boards, and semi custom due to that its connections inter FPGAs as well as connections to external interfaces are user-defined and tailored for a specific design. Nevertheless, all the inter-FPGA connections are realized using cables and connectors instead of PCB traces.

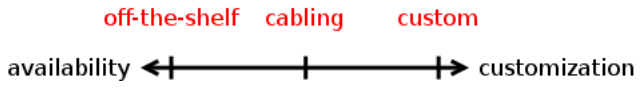


Figure 1.4: *Three different multi-FPGA prototyping platforms*

1.2 Problems

1.2.1 Pin Limitation Problem

Mapping multi-million gates SoCs into any kind of the multi-FPGA platform is very challenging. The mapping process can be divided in two main steps. The first step is the partitioning of the design, meaning that the design is divided into several parts. Each part fits in the logic capacity of a single FPGA. The signals crossing design's parts located in different FPGAs are called cut nets. The second step routes the cut nets, meaning that a cut net is allocated to an inter-FPGA track on the platform. One should note that even though the logic capacity and the number of FPGA I/Os are increasing generation after generation, the logic capacity increases at a much higher rate than the number of FPGA I/Os [Schirrmeister, 2014]. Figure 1.5 shows the ratio between the logic capacity and the number of I/Os for each FPGA generation from Virtex-4 to Virtex-7 in Xilinx

Virtex Family and from Stratix2 to Stratix5 in Altera Stratix Family [DINI, 2014]. About 2000 gates are trying to get through one FPGA I/O in Virtex-4 or Stratix2. In the case of the latest generation Virtex-7 (resp. Stratix5), this is almost 20000 (resp. 16000) gates. This means that FPGA I/Os are becoming a scarce resource, worsening the inter-FPGA bandwidth generation after generation. Unfortunately, multi-FPGA platforms suffer from large timing delays in inter-FPGA communication compared to intra-FPGA net delays [Amos et al., 2011]. Therefore, this trend has a direct impact on multi-FPGA system performance (in terms of the system clock frequency) and makes it more and more difficult to prototype an SoC/ASIC design at proper performance.

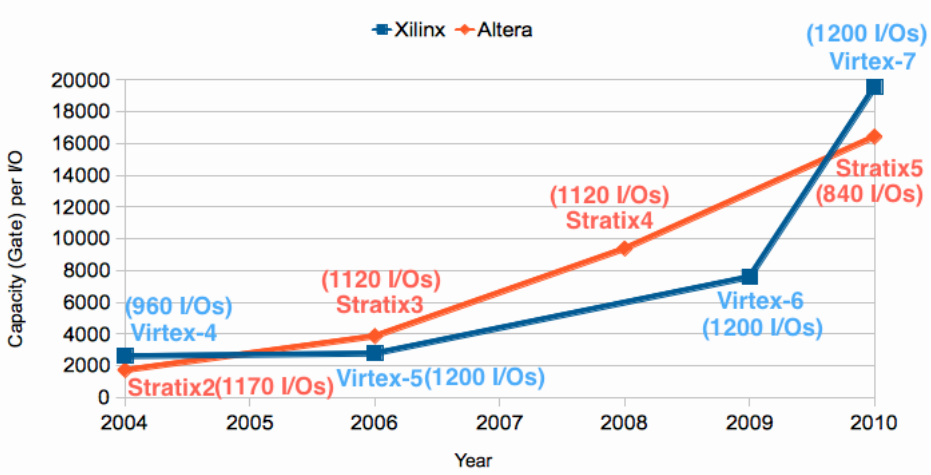


Figure 1.5: *The ratio FPGA logic capacity over I/Os*

1.2.2 Problem of Crafting a Custom Platform

70% of multi-FPGA prototyping platforms are home-made custom platforms due to performance requirement, external interfaces, and cost. Indeed home-made custom platforms are tailored for a specific design and external interfaces, and the cost decreases as more platforms are needed. Nevertheless, crafting a home-made custom multi-FPGA platform is today a manual process. Therefore, three critical aspects exist:

- Productivity: Crafting a home-made custom multi-FPGA platform is a time-consuming process (about 9 months [Sekhar, 2014]). The performance and the cost of the platform lie on the FPGA expertise and SoC DUT knowledge of the prototyping team.
- Exploration: There are many different FPGA types (i.e. vendor: Xilinx [Xilinx, 2014] or Altera [Altera, 2014], family: Virtex-7 or Stratix5, device: 2000T or GXAB, and package: FLG1925 or F1932), and different FPGA types have different logic capacity and numbers of FPGA I/Os. For a given design, the FPGA type chosen by the engineers influences the achieved performance and cost. The tradeoff between the performance and the cost exists due to that the performance of multi-FPGA platforms is limited by the inter-FPGA

communications [Amos et al., 2011]. The board exploration with different FPGA types, which helps the engineers to design an optimum prototyping platform, can not be done.

- Performance: As the ratio between the logic capacity and the number of FPGA I/Os is increasing at an exponential rate, it becomes more and more challenging to keep performance as it was in multi-FPGA platform.

1.2.3 Problem of Cabling Paradigm

The cabling platform benefits from the availability and the customization. The performance of the cabling platform depends on the distribution of the cables and the placement of the external interfaces. Nevertheless, there is no tool to automatically have a solution for the cable distribution. Today, the cables (resp. the external interfaces) are distributed (resp. placed) according to the experience of board designers. A high-performance cables distribution becomes more and more difficult to be achieved due to the pin limitation.

Compared to the off-the-shelf platform, the added value, in terms of performance, of cabling or custom platforms can be heavily impaired by an inefficient board design.

1.3 Thesis Contributions

In order to solve the problems presented above, the major contributions of the manuscript include the following:

- A new routing algorithm is proposed to spare FPGA I/Os by exploiting multi-point tracks that connect more than two FPGAs, thus increasing the performance.

- An automatic design flow for creating a custom platform is proposed, thus increasing the productivity, enabling the board exploration, and optimizing cost and performance.

- The cabling platform is proposed where one board is composed of one FPGA and several connectors. The connections inter FPGAs as well as the connections to external interfaces can be added or removed by only connecting or disconnecting the cables (resp. daughter boards) with or from the connectors. Then, an algorithm is proposed to automatically find a solution for the cable distribution.

- Thanks to the developed automatic tools, the three different categories of multi-FPGA platforms (off-the-shelf, custom, and cabling) are compared. The performance gains between these platforms are quantified.

1.4 Thesis Organization

The rest of this manuscript is organized as follows.

Chapter 2 studies the state of the art of multi-FPGA prototyping platforms and details the inter-FPGA communication architectures. As there are three different multi-FPGA prototyping platforms, the state of the art is classified into four parts: Hardwired off-the-shelf multi-FPGA platform, Hardwired custom multi-FPGA platform, Cabling multi-FPGA platform, and Comparison of different platforms. Then, the inter-FPGA communication architectures are discussed due to that they are the critical path of all the multi-FPGA prototyping platforms.

Chapter 3 focuses on the off-the-shelf platform. An overview of multi-FPGA off-the-shelf platforms and the implementation flow are presented. There are two types of inter-FPGA tracks: 2-point tracks that connect two FPGAs and multi-point tracks that connect more than two FPGAs. A new routing algorithm of routing cut nets in 2- and multi-point tracks is proposed.

Chapter 4 focuses on the custom platform. First, an overview of the custom platform is presented. Then, the automatic design flow for creating a custom platform is proposed. Different steps of the proposed automatic design flow are explored in order to achieve lower cost and higher performance. With the proposed automatic design flow, board exploration (exploring different FPGA types for the given design) has been addressed. Plenty of feasible solutions can be generated in board exploration and board designers can make the tradeoff between cost and performance.

Chapter 5 focuses on the cabling platform. A cabling platform is proposed with an algorithm to automatically find a solution for the cable distribution.

Chapter 6 compares the achieved performances for a set of designs mapped on the three different categories of multi-FPGA platforms. The performance gains between these platforms are quantified.

Finally, Chapter 7 concludes the manuscript and suggests topics for future work.

Chapter 2

Background

2.1 Introduction

The previous Chapter has presented the different multi-FPGA prototyping platforms and the difficulties using them optimally. The purpose of this Chapter is to give to the user the necessary background to understand the rest of this manuscript. This Chapter is organized as follows:

- Section 2.2 discusses the state of the art of multi-FPGA prototyping platforms.
- Section 2.3 details the inter-FPGA communication architectures, which are the critical path of all the multi-FPGA prototyping platforms in performance (in terms of the system clock frequency).

2.2 State of the Art

Despite the fact that hardware/software integration is a more and more concerning problem, few papers have been published regarding multi-FPGA prototyping platforms. As there are three different multi-FPGA prototyping platforms, the study of the related work is classified into four parts:

- Hardwired off-the-shelf multi-FPGA platform
- Hardwired custom multi-FPGA platform
- Cabling multi-FPGA platform
- Comparison of different platforms

2.2.1 Hardwired Off-the-Shelf Multi-FPGA Platform

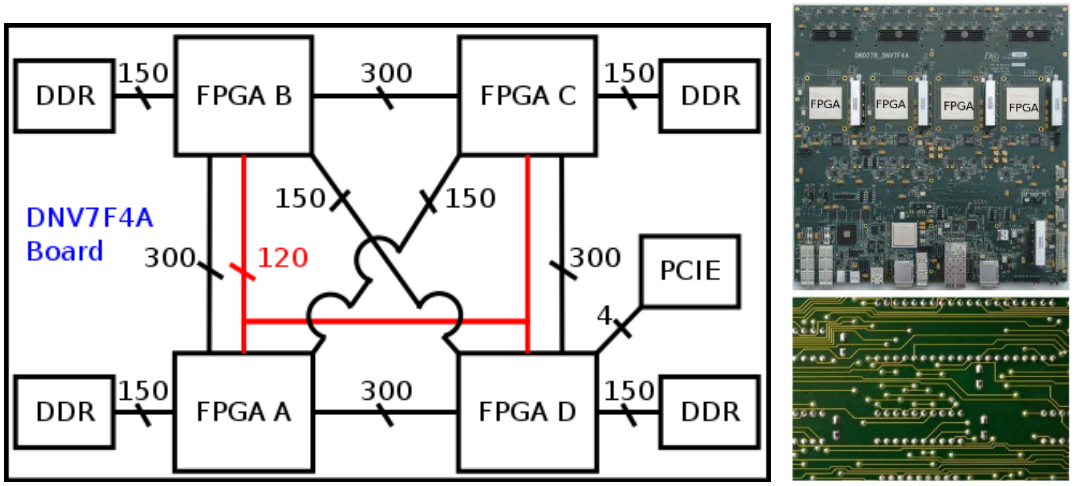
The related work of the hardwired off-the-shelf multi-FPGA platform contains two parts:

- The platform overview that presents the definition of the off-the-shelf platform
- The state of the art in the implementation flows, more specifically the state of the art in the routing algorithms

2.2.1.1 Platform Overview

In this manuscript, the off-the-shelf platform consists of a ready-made generic multi-FPGA board, where all the inter-FPGA connections are fixed and realized using PCB traces. The connections to external interfaces are also fixed but can be realized using PCB traces or connectors (connected to daughter boards). An example of such platform is the commercial platform DNV7F4A as shown in Figure 2.1, which is the latest platform from Dini Group [DINI, 2014]. The platform uses four FPGAs of the latest Xilinx FPGA (family: Virtex-7, device: 2000T, and package: FLG1925, thus FPGA type: XC7V2000TFLG1925). Each FPGA is connected with a DDR by connectors. Only the FPGA D is connected with a PCIE by connectors. The connectors are fixed and specific. This means that the connectors to one type of the external interface (i.e. DDR) can not be transferred to the connectors to another type of the external interface (i.e. PCIE). If the prototyped design has less than 4 DDRs, several FPGA I/Os connected with DDRs are wasted. If the prototyped design has a PCIE, the PCIE part of the prototyped design is forced to be placed in FPGA D. These constraints add to the complexity of the design partitioning and may reduce the performance. The number of inter-FPGA tracks is shown in Figure 2.1(a). All the inter-FPGA tracks are fixed. There are two types of inter-FPGA tracks in the multi-FPGA prototyping platform: 2-point tracks and multi-point tracks. The number of multi-point tracks in the example is 120. The number of 2-point tracks between a pair of FPGAs are generic and balanced: 300 connections in all the horizontal and vertical pairs, and 150 connections in all the diagonal pairs. When implementing a specific

design, these generic and balanced connections constraint the performance. Figure 2.1(b) shows the picture of the platform and the PCB traces.



(a) The architecture (b) The picture

Figure 2.1: *The off-the-shelf multi-FPGA platform*

Major existing commercial off-the-shelf platforms are Cadence Protium Rapid Prototyping Platform [Protium, 2014], Dini Group [DINI, 2014], S2C [S2C, 2014], and HyperSilicon [HyperSilicon, 2014].

2.2.1.2 Implementation Flow of Multi-FPGA Platforms

Implementing multi-million gates SoCs into multi-FPGA platforms is very challenging. The implementation flow as shown in Figure 2.2, which starts by inputting the design RTL, can be divided in three main steps:

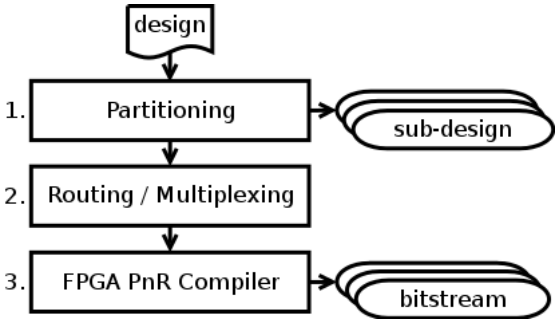


Figure 2.2: *The typical implementation flow*

1. Partitioning: As the SoC/ASIC design is bigger than the FPGA, the design is partitioned into several parts according to the number of FPGAs in the platform. Each part's capacity fits in a single FPGA. The signals crossing design's parts located in different FPGAs are called cut nets.

2. Routing / Multiplexing: The cut nets are routed meaning that a cut net is allocated to an inter-FPGA track in the platform. As there are fewer available inter-FPGA tracks than the number of cut nets, several cut nets need to be multiplexed and sent together onto a single track on the platform.

3. FPGA PnR Compiler: The bitstream for each FPGA is generated from the sub-design. Then, the generated bitstreams are downloaded into the platform to model the design.

Nowadays, there are four commercial implementation flow tools: Cadence Protium tool [Protium, 2014], Synopsys ProtoCompiler tool [ProtoCompiler, 2014], Auspy tool [Auspy, 2014], S2C tool [S2C, 2014], and Flexras Wasga tool [Flexras, 2014], targeting at automatizing the implementation flow.

This manuscript will study the process of the routing and propose an improved routing algorithm. The state of the art of the routing algorithms is discussed in the following. Routing algorithms for multi-FPGA platforms heavily depend on the underlying hardware routing architectures. Thus, they can be classified according to the routing architectures.

Direct architecture

All the FPGAs are directly interconnected [Walters, 1991] as shown in Figure 2.3. These direct inter-FPGA tracks can be divided into two categories: 2-point tracks directly connect one FPGA to another, and multi-point tracks connect several FPGAs together.

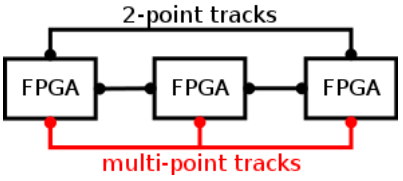


Figure 2.3: *The direct architecture*

An iterative, congestion-aware routing algorithm is provided by [Turki et al., 2013] as shown in Figure 2.4. Nevertheless, the algorithm only uses 2-point tracks and routes multi-terminal nets through a sequence of 2-point tracks. The routing algorithm is detailed as follows.

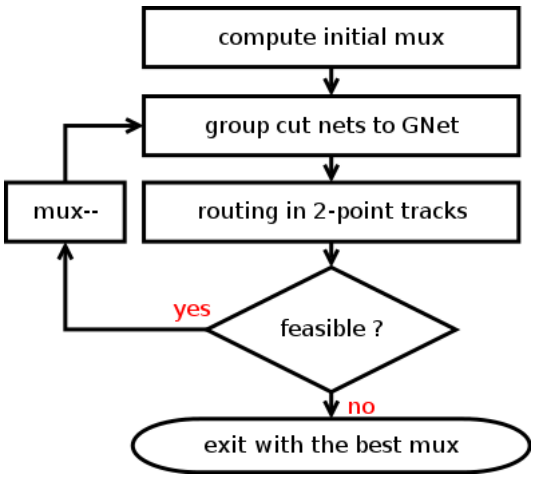


Figure 2.4: *The proposed routing algorithm*

- Compute the initial maximum number of cut nets passing through one multiplexer (noted as mux).

  After loading the partitioned design and the multi-FPGA platform architecture, the algorithm will calculate initial mux by obtaining the maximum ratio of cut nets and inter-FPGA tracks between each pair of FPGAs.

- Group cut nets to GNet

  The cut nets, which have the same driver FPGA and the same receiver FPGAs, are grouped together. The group, which is called GNet, will be put in one track. Each GNet contains a maximum of mux cut nets.

- Routing in 2-point tracks

  PathFinder [McMurchie and Ebeling, 1995] was used primarily for routing intra-FPGA nets. It is adapted to deal with GNets and routes GNets in 2-point tracks. An iterative negotiation-based approach is used in PathFinder. During the first routing iteration, GNets are freely routed without paying attention to track sharing. Individual GNets are routed using Dijkstra's shortest path algorithm [Cormen et al., 2009]. At the end of the first iteration, tracks may be congested because multiple GNets try to use one track. During subsequent iterations, the cost of using a track is increased, based on the number of GNets that share the track, and the history of congestion on that track. Thus, GNets are forced to negotiate for tracks. If a track is highly congested, GNets which can use lower congestion alternatives are forced to do so. On the other hand, if the alternatives are more congested than the track, then a GNet may still use that track. Due to that one track can only contain one GNet, if there are two GNets in one track in the result of PathFinder, there is a conflict. A routing result is feasible only when the number of conflicts is 0.

- The proposed algorithm tries to minimize mux. If feasible, decrease the mux and iterate the previous steps beginning from grouping cut nets with the new mux. If not feasible, exit with the best mux.

Nevertheless, this algorithm does not exploit multi-point tracks, which may waste FPGA I/Os.

Indirect architecture

All the FPGAs are interconnected through partial crossbars [Varghese et al., 1993] as shown in Figure 2.5.

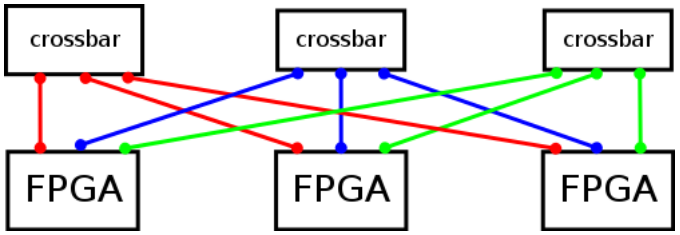


Figure 2.5: *The indirect architecture*

Multi-terminal nets routing, through partial crossbars, is discussed in [Mak and Wong, 1997] [Song et al., 2003] [Ejnioui and Ranganathan, 2003]. Even though multi-terminal nets routing on indirect architectures spares FPGA I/Os, multi-FPGA platforms performance is tied to inter-FPGA delays, and this kind of architecture worsens this effect as nets are routed through at least 2 inter-FPGA tracks plus the crossbar component. Moreover, if most of cut nets are 2-terminal nets, this architecture will waste FPGA I/Os. Therefore, this architecture is not used for FPGA-based prototyping.

Hybrid architecture

A mixture of direct connections and partial crossbars are used in [Khalid, 1999] [Khalid and Rose, 2000] as shown in Figure 2.6.

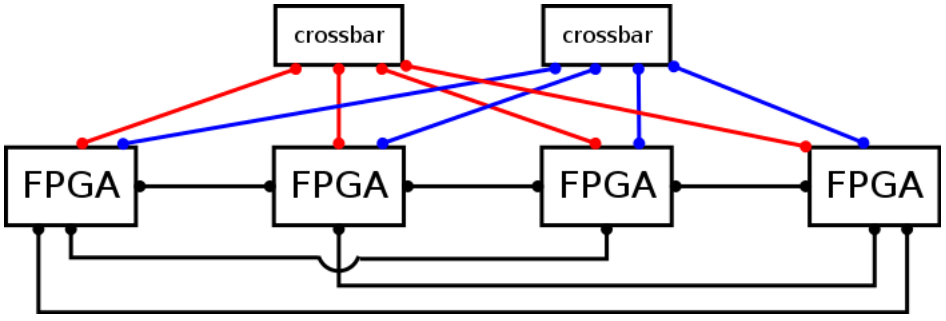


Figure 2.6: *The hybrid architecture*

The routing algorithm in [Jain et al., 2002] proposes the use of a crossbar to route multi-terminal nets and direct connections to route 2-terminal nets. However, this kind of architecture suffers of the same performance degradation as the indirect architecture and is not used for FPGA-based prototyping.

Conclusion

Industrial multi-FPGA prototyping platforms (Cadence Protium Platform [Protium, 2014], Dini Group Platform [DINI, 2014] and Reflex FPP Platform [ReFLEX, 2014]) have adopted direct architecture for performance motivation. They afford 2- and multi-point tracks. Nevertheless, the existing tools do not automatically route and multiplex cut nets in multi-point tracks, which waste FPGA I/Os that are already a scarce resource. In Chapter 3, which focuses on the off-the-shelf platform, we will target the direct architecture and we propose to spare FPGA I/Os by automatically routing and multiplexing multi-terminal nets over multi-point tracks.

2.2.2 Hardwired Custom Multi-FPGA Platform

The related work of the hardwired custom multi-FPGA platform contains two parts:

- The platform overview that presents the definition of the custom platform
- The state of the art of the design flows for creating a custom platform

2.2.2.1 Platform Overview

The custom platform consists of a build-your-own specific multi-FPGA board, where all the inter-FPGA connections are realized using PCB traces as well. The advantage of the custom platform is that it is tailored for a specific design. An example is shown in Figure 2.7. The design is bigger than the largest FPGA available and three of them are needed. There are many signals crossing from Part A to Part B but only few from Part A to Part S. Therefore, more inter-FPGA tracks are allocated between FPGA A and FPGA B than between FPGA A and FPGA S. In terms of external interfaces, there is only 1 DDR in the example. Therefore, 1 DDR is reserved only for FPGA B that contains the memory controller.

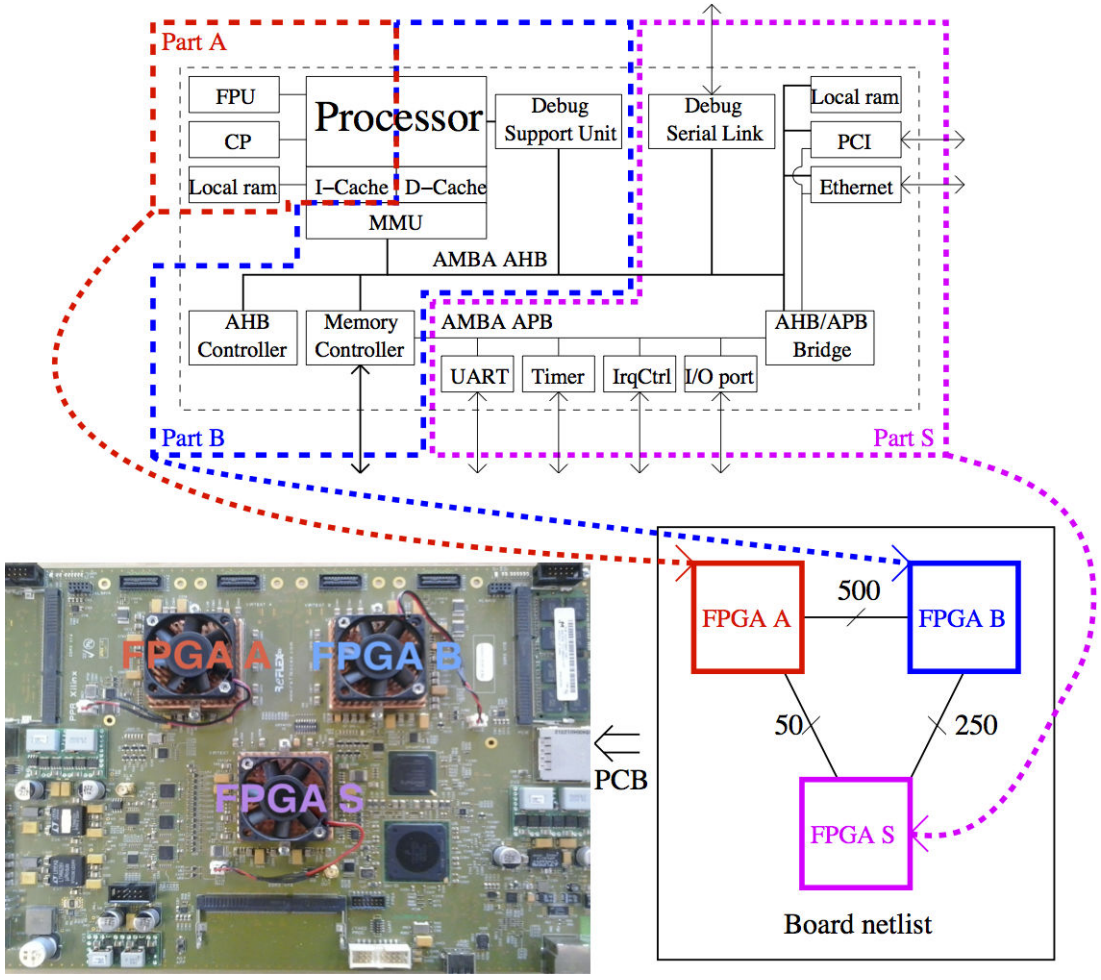


Figure 2.7: *Create a Custom Platform for a specific design*

Different from the off-the-shelf platform, which has generic and balanced connections as shown in Figure 2.1, tracks of the custom platform are user-defined and tailored for the given design to achieve higher performance. Different from the off-the-shelf platform, which has 1 DDR reserved for each FPGA as shown in Figure 2.1 and wastes FPGA I/Os due to that there is only 1 DDR in the example, the tracks to external interfaces of the custom platform are tailored for the given design and no FPGA I/O is wasted. Therefore, the design partitioning is facilitated due to that the DDR is connected to the FPGA where the memory controller logic is placed in the custom platform, thus higher performance can be achieved. Nevertheless, 4-7 months are needed for the PCB layout and the PCB fabrication.

2.2.2.2 Design Flow for Creating a Custom Platform

Typical design flow

The typical design flow as shown in Figure 2.8 for creating a home-made custom multi-FPGA platform is detailed as follows:

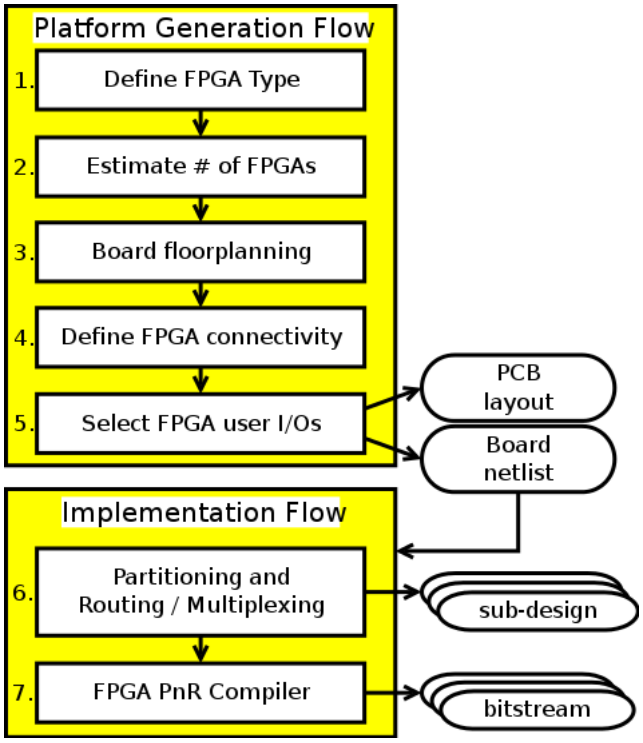


Figure 2.8: *The typical design flow for creating a custom platform*

1. Choose the FPGA type (i.e. vendor: Xilinx [Xilinx, 2014] or Altera [Altera, 2014], family: Virtex-7 or Stratix5, device: 2000T or GXAB, and package: FLG1925 or F1932).

2. Estimate the minimum number of FPGAs required in the platform (according to the logic capacity of the input design, the logic capacity of the chosen FPGA type, and the maximum FPGA logic capacity utilization).

3. Floorplan FPGAs and external interfaces on the Printed Circuit Board (PCB).

4. Define the FPGA connectivity (FPGA-to-External-Interface and inter-FPGA tracks).

5. Select FPGA I/Os. The I/O DRC rules (such as clock capability, clock region and I/O standard voltage reference levels) are considered, when choosing I/Os for each track. Then, a board netlist and a PCB layout are generated.

6. Map the SoC/ASIC design on this latter board, meaning that the design is partitioned and routed into each FPGA. As there are fewer available inter-FPGA tracks than the number of cut nets, several cut nets need to be multiplexed and sent together onto a single track on the platform.

7. Run the PnR Compiler with the design sub-netlists to generate the bitstreams for each FPGA. Then, the generated bitstreams are downloaded into the platform to model the design.

The typical design flow is mainly manual and iterative. Therefore, this process is time-consuming, and the optimization of the cost and the performance depends on the designers' experience.

Cadence design flow

In [Cadence, 2011], a design flow for creating a custom platform is proposed, but only the process of selecting FPGA user I/Os is automated (by the Cadence FSP tool [FSP, 2014]) as shown in Figure 2.9.

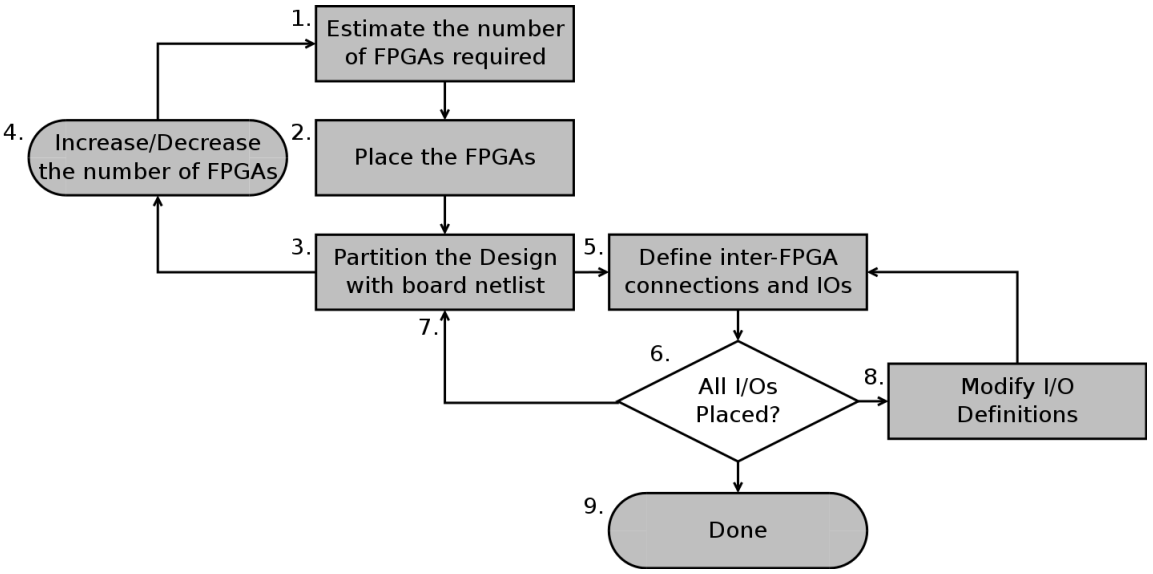


Figure 2.9: *Cadence's design flow for creating a custom platform.*

1. Estimate the minimum number of FPGAs required for partitioning the design RTL according to the chosen FPGA type.
2. Place the FPGAs on the Allegro FPGA System Planner canvas [FSP, 2014] and generate a Verilog board netlist with the FPGAs,
3. Use a partitioning tool to ensure that the RTL can be partitioned within the selected number of FPGAs.
4. If the selected number of FPGAs is not reasonable, increase or decrease the number of FPGAs and iterate the previous processes.
5. Use the top level of the partitioned RTL to define the connections required among the different FPGAs.
6. Once the I/O synthesis is completed in Allegro FPGA System Planner, the Verilog board connectivity file can be generated. Some iterations with the I/O definitions may be needed to ensure that all the I/Os can be placed on the different FPGAs.

7. Use this connectivity file once again in the RTL partitioning tool. The RTL partitioning tool can now map the wires in the top level to ports/traces on the board

8. In some cases, it may be required to iterate a bit to get to successful results, analyze results of the partitioning tools and make the required changes to the FPGA board architecture.

9. Once all the requirements are met, the board layout can be generated.

Even though the proposed point tool Cadence FSP automatically selects FPGA user I/Os, the Cadence design flow for creating a custom platform is still mainly manual and iterative. Therefore, this process is time-consuming, and the optimization of the cost and the performance depends on the designers' experience.

Conclusion

The existing design flows are mainly manual and iterative, thus creating a custom platform is time-consuming and the optimization of the cost and the performance depends on the designers' experience.

In Chapter 4, which focuses on the custom platform, we propose an automatic design flow for creating a custom multi-FPGA platform. The proposed automatic design flow reduces the time-to-market of new products and lowers the entry barrier of board designers while optimizing the cost and the performance. With the proposed automatic design flow, board exploration (exploring different FPGA types for the given design) has been addressed. Plenty of feasible solutions can be generated in board exploration and board designers can make the tradeoff between cost and performance.

2.2.3 Cabling Multi-FPGA Platform

The related work of the cabling multi-FPGA platform contains two parts:

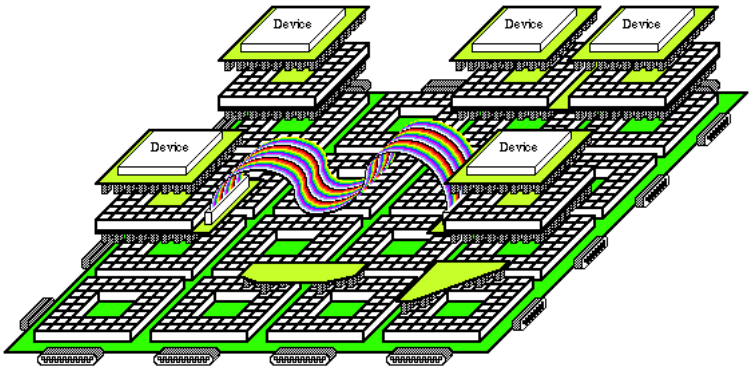
- The platform overview that presents the definition of the cabling platform
- The state of the art of the design flows for creating a cabling platform

2.2.3.1 Platform overview

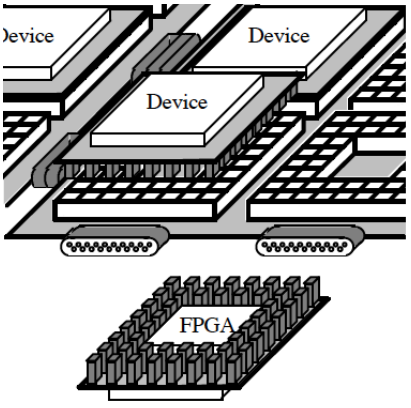
The cabling platform, which is a relatively new notion compared to other two platforms, consists of multiple ready-made FPGA boards connected by cables and connectors. In between the off-the-shelf and the custom platform, the cabling platform is semi off-the-shelf as it is ready-made, and semi custom as the inter-FPGA connections can be changed by connecting or disconnecting the cables in order to be tailored for the given design.

The hybrid cabling platform

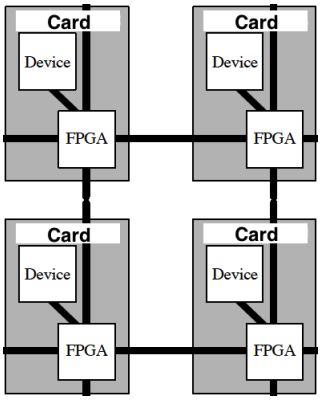
In [Hauck, 1995], a hybrid cabling platform as shown in Figure 2.10(a) is proposed. The inter-FPGA tracks are realized by a hybrid of the cables and the PCB traces. To allow a specific circuit to be implemented in this structure, the SpringBok system is composed of a baseplate with sites for cards. The cards in the platform as shown in Figure 2.10(b) are large enough to contain an arbitrary device on the top, as well as an FPGA on the bottom. Note that the device can be a chip, or external interfaces (such as PCI, DDR). The inter card routing structure is a mesh as shown in Figure 2.10(c) and the connections are fixed. There is the potential that the simple connections will not be able to accommodate all the logic or routing assigned to a given location (Intermediate FPGAs are needed from the driver FPGA to the destination FPGA, which degrades the performance). However, as opposed to the off-the-shelf platform, "extender" cards can be inserted between a card and the baseplate to deal with these problems. For signals that must go long distances in the array, sets of "extender" cards with cables can be inserted to carry these long-distance wires. Nevertheless, the inter-FPGA tracks in the hybrid cabling platform is not fully customized for the given design due to that several inter-FPGA tracks are realized by fixed PCB traces. Therefore, the achieved performance is limited.



(a) SpringBok



(b) Two-sided



(c) Interconnection pattern

Figure 2.10: *Multi-FPGA Platform: SpringBok*

The quasi cabling platform

In [Kulmala et al., 2007] and IBM [Asaad et al., 2012], a cabling platform is used as shown in Figure 2.11. All the inter-FPGA tracks are realized by cables, but the connections to external interfaces are realized using PCB traces. The inter-FPGA tracks can be changed in order to be tailored for the given design. Due to that the cabling platform is ready-made, the connections to external interfaces may be not tailored for the given design. Thus, FPGA I/Os occupied by external interfaces may be wasted. Moreover, there is no discussion about the cable distribution. The cable distribution and the external interface placement, which are essential to the achieved performance in the cabling platform, depend on the experience of board designers.

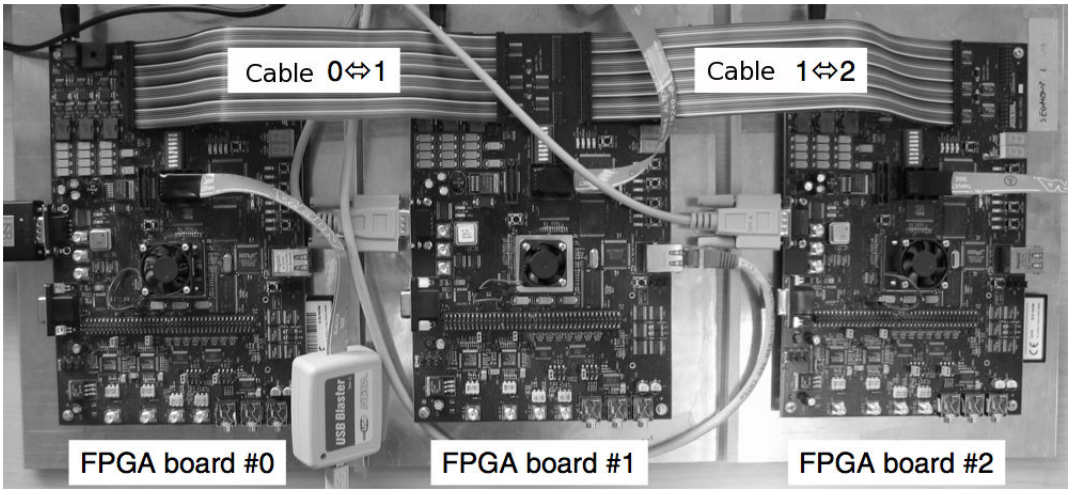


Figure 2.11: *A photo of Kulmala's quasi cabling platform*

The cabling platform

The cabling platform consists of multiple ready-made FPGA boards connected by cables and connectors. Almost all the FPGA I/Os are used for FPGA-to-connector connections. The connections between FPGAs as well as the connections to external interfaces can be added or removed by only connecting or disconnecting the cables (resp. daughter boards) with or from the connectors in order to be tailored for the given design.

The proFPGA cabling platform as shown in Figure 2.12 is proposed in [Prodesign, 2014]. For each FPGA, there are only 4 connectors. The width of connectors is defined as the pairs of tracks passing through this connector. Among the four connectors, three of them have 148 pairs of tracks, and the rest one has 98 pairs of tracks. Nevertheless, the coarse-grained connector may waste FPGA I/Os when external interfaces do not occupy so many FPGA I/Os but need at least one connector. Moreover, there is no tool to automatically have a solution for the cable distribution. The cable distribution and the external interface placement, which are essential to the achieved performance in the cabling platform, depend on the experience of board designers.

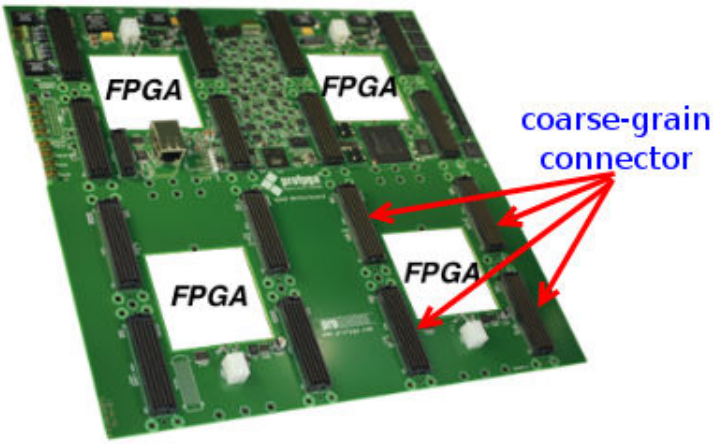


Figure 2.12: *proFPGA*

The HAPS cabling platform as shown in Figure 2.13 is proposed by Synopsys [HAPS, 2014]. For each FPGA, there are 23 connectors. Each connector has 24 pairs of tracks. Comparing to proFPGA, the connector in HAPS is fine-grained. For example, the external interface DDR3 that occupies 72 pairs of tracks, covers 3 connectors in HAPS. Therefore, no FPGA I/O is wasted. In proFPGA, this DDR3 covers the connector with 98 pairs of tracks and 26 pairs of FPGA I/Os are wasted. Again, there is no tool to automatically have a solution for the cable distribution. Therefore, the cable distribution and the external interface placement, which are essential to get higher performance in the cabling platform, depend on the experience of board designers.

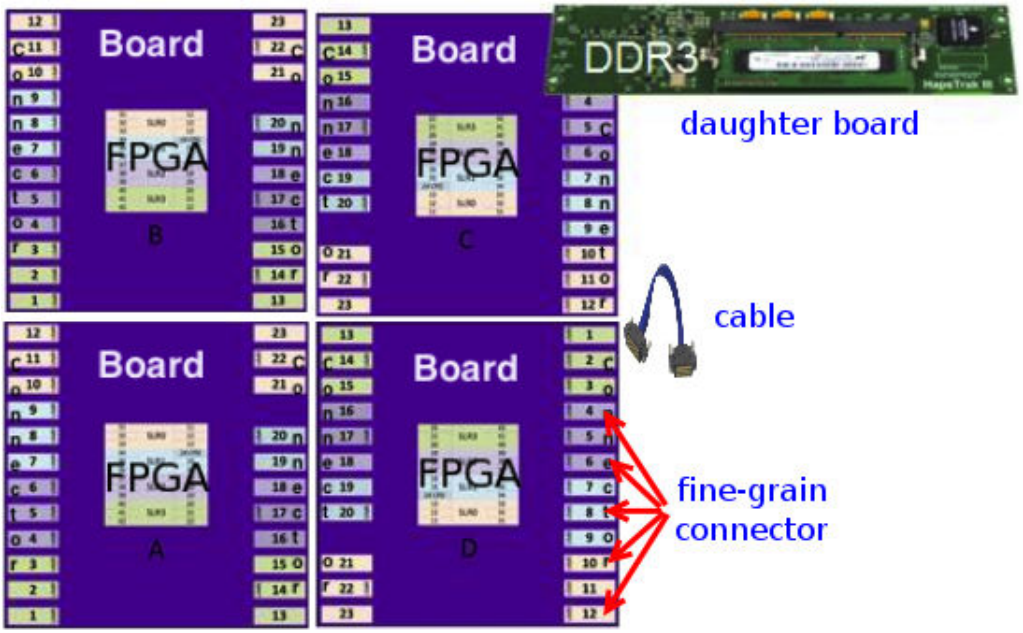


Figure 2.13: *Synopsys HAPS*

2.2.3.2 Design Flow for Creating a Cabling Platform

The typical design flow as shown in Figure 2.14 for creating a cabling multi-FPGA platform is detailed as follows:

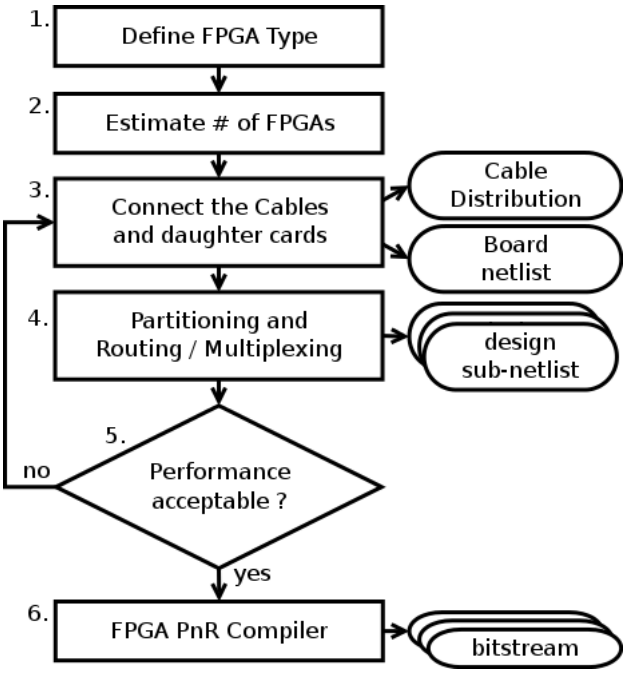


Figure 2.14: *The typical design flow for creating a cabling platform*

1. Choose the FPGA type (i.e. vendor: Xilinx [Xilinx, 2014] or Altera [Altera, 2014], family: Virtex-7 or Stratix5, device: 2000T or GXAB, and package: FLG1925 or F1932).

2. Estimate the minimum number of FPGAs required in the platform (according to the logic capacity of the input design, the logic capacity of the chosen FPGA type, and the maximum FPGA logic capacity utilization.

3. Manually find a solution of the distribution of the cables and the external interface daughter boards. Then, the cables and external interface daughter boards are connected. Finally, a board netlist is generated.

4. Map the SoC/ASIC design on this latter board, meaning that the design is partitioned and routed into each FPGA. As there are fewer available inter-FPGA tracks than the number of cut nets, several cut nets need to be multiplexed and sent together onto a single track on the platform.

5. Evaluate the performance. If the performance is acceptable, pass to the next step. If not, iterate the previous step from re-connecting the cables and external interface daughter boards.

6. Run the PnR Compiler with the design sub-netlists to generate the bitstreams for each FPGA. Then, the generated bitstreams are downloaded into the platform to model the design.

The typical design flow is mainly manual and iterative. Therefore, this process is time-consuming, and the optimization of the performance depends on the designers' experience.

2.2.3.3 Conclusion

According to our knowledge, there is no tool to automatically have a solution for the cable distribution. In Chapter 5, which focuses on the cabling platform, we propose a cabling platform with an algorithm to automatically optimize the cable distribution and the external interface placement. Nowadays, the two existing cabling platforms (proFPGA and Synopsys HAPS) have different width granularity of the connectors (resp. cables). With the proposed algorithm, the optimal width of connectors in terms of performance is explored.

2.2.4 Comparison of Different Multi-FPGA Platforms

Combining [Amos et al., 2011] [Cadence, 2011], these three different platforms are compared qualitatively in terms of availability, performance, flexibility, and cost as shown in Table 2.1.

Table 2.1: Comparison of Different Multi-FPGA Prototyping Platforms

Multi-FPGA Prototyping Platforms	Availability	Performance		Flexibility	Cost	
		Inter-FPGA data rate	tracks distribution		Unit Price	Deployment Cost
Hardwired Off-the-Shelf	instantaneous	high	generic & balanced	medium	low	high
Cabling	1-2 months	high	designers' experience	high	low	high
Hardwired Custom	4-7 months	designers' experience	designers' experience	low	high	low

The availability is the time to get access to the platform. The off-the-shelf platform is available instantly. For the cabling platform, 1-2 months are needed to find an optimal cable distribution. For the custom platform, 4-7 months are needed to find a good solution due to that the PCB layout and the PCB fabrication is needed. The achieved performance of multi-FPGA platforms depends on the inter-FPGA data rate and inter-FPGA tracks distribution. For the off-the-shelf and the cabling platform, the data rate is high due to that they are ready-made by the FPGA experts in the commercial companies. For the custom platform done by the in-house team, it depends on designers' experience on how to design high-speed signaling, high performance multiplexing and so forth. For the off-the-shelf platform, the inter-FPGA tracks distribution is generic and balanced. For the cabling and the custom done by the in-house team, it depends on the designers' experience on how to distribute inter-FPGA tracks in order to achieve higher performance. Naturally, the custom platforms should achieve highest performance because it is tailored for the given design.

Unfortunately, the performance of the cabling and the custom platform heavily depend on designers' experience because designing such platforms is still a manual process. Therefore, not so good performance platforms can be often seen. The flexibility of the platform is its ability to be tailored for different designs. The cabling platform has the highest flexibility due to that the same platform can be tailored for different designs by only re-connecting the cables. The off-the-shelf platform has medium flexibility due to its generic and balanced connections. The custom platform has the lowest flexibility due to that one custom platform tailored for one design may be not tailored for another design. The unit price for the custom platform is high as it integrates the development cost. The deployment cost corresponds to the quantity of platforms needed. This cost for custom platforms is low because it consists of the Bill-Of-Material (BOM) plus the production cost.

2.2.5 Conclusion of the State of the Art

Different kinds of platforms are widely used: off-the-shelf [Hyder and Wawrzynek, 2005], custom [Krupnova, 2004], and cabling [Kulmala et al., 2007] [Asaad et al., 2012]. The contributions of this manuscript are classified into four parts: Firstly, a new routing algorithm is proposed to spare FPGA I/Os by exploiting multi-point tracks. Secondly, an automatic design flow for creating a custom platform is proposed. Thirdly, the cabling platform and an algorithm to automatically optimize the cable distribution and the external interface placement, are proposed. Finally, the achieved performances are compared when a set of designs are mapped on the three platforms. The performance gains between these platforms are quantified.

Even though the three platforms are different, they all have their performance limited by the inter-FPGA connections. Different inter-FPGA communication architectures, more or less complex, can be implemented leading to different achieved performances. Before exploiting different multi-FPGA platforms, we have to explain these different architectures.

2.3 Inter-FPGA Communication Architectures

According to [Amos et al., 2011] [Pinmux, 2014], inter-FPGA tracks are the critical path of all the multi-FPGA prototyping platforms in performance. Therefore, inter-FPGA communication architectures need to be detailed in order to understand the rest of this manuscript.

2.3.1 Time-Division-Multiplexing

As FPGA I/O pins are becoming a scarce resource, the number of cut nets outmatches the few available inter-FPGA physical tracks. Therefore, the cut nets are sent between FPGAs in a pipelined way [Babb et al., 1997] [Inagi et al., 2010] using the Time-Division-Multiplexing (TDM) technique. This technique consists in multiplexing several cut nets onto a single track on the platform. The maximum number of cut nets passing through one FPGA I/O is called the TDM ratio. Due to the pin limitation, the TDM ratio is increasing generation after generation. In [Krupnova, 2004], four ST Microelectronics SoCs are prototyped on a platform made of Virtex-II FPGA, and the TDM ratio is 4. In [Schelle et al., 2010], the Intel Nehalem processor core is mapped on a platform mixing Virtex-4 and Virtex-5, and the TDM ratio is 24. In [Asaad et al., 2012], the IBM's Bluegene/Q is prototyped on a Virtex-5 only platform, and the TDM ratio achieved is between 32 and 96.

The architecture of the basic Time-Division-Multiplexing is shown in Figure 2.15. The cut nets go through, a multiplexer (noted as MUX) connected to the FPGA I/O(s), the inter-FPGA track, and are de-multiplexed (noted as $DMUX$) in the receiving FPGA. After the partitioning, the parts located in different FPGAs need to work in the same system clock (noted as sys_clk). The multiplexer and the de-multiplexer use a fast clock (noted as if_clk) to propagate the multiplexed data over the inter-FPGA track. Each cut net has a time slot allocated corresponding to a fast clock cycle. All the multiplexed data should be sent in-between two system clock pulses. The latency is the number of the fast clock cycles from the first capture to the last update. In the multi-FPGA based prototyping, the system clock frequency is the fast clock frequency divided by the latency and need to be synchronous with the fast clock frequency [Amos et al., 2011]. Therefore, for a given inter-FPGA communication architecture and a given TDM ratio, the total latency need to be fixed. Ideally, when the latencies of added MUX and DMUX IPs are not taken into consideration, the latency equals the TDM ratio plus one. For example, in Figure 2.15, if the TDM ratio is 4, ideally the latency will be 5. For a given inter-FPGA communication architecture, the higher is the TDM ratio, the higher is the latency and the lower is the system clock frequency. Generation after generation the TDM ratio is increasing, implying that multi-FPGA system performance is reducing.

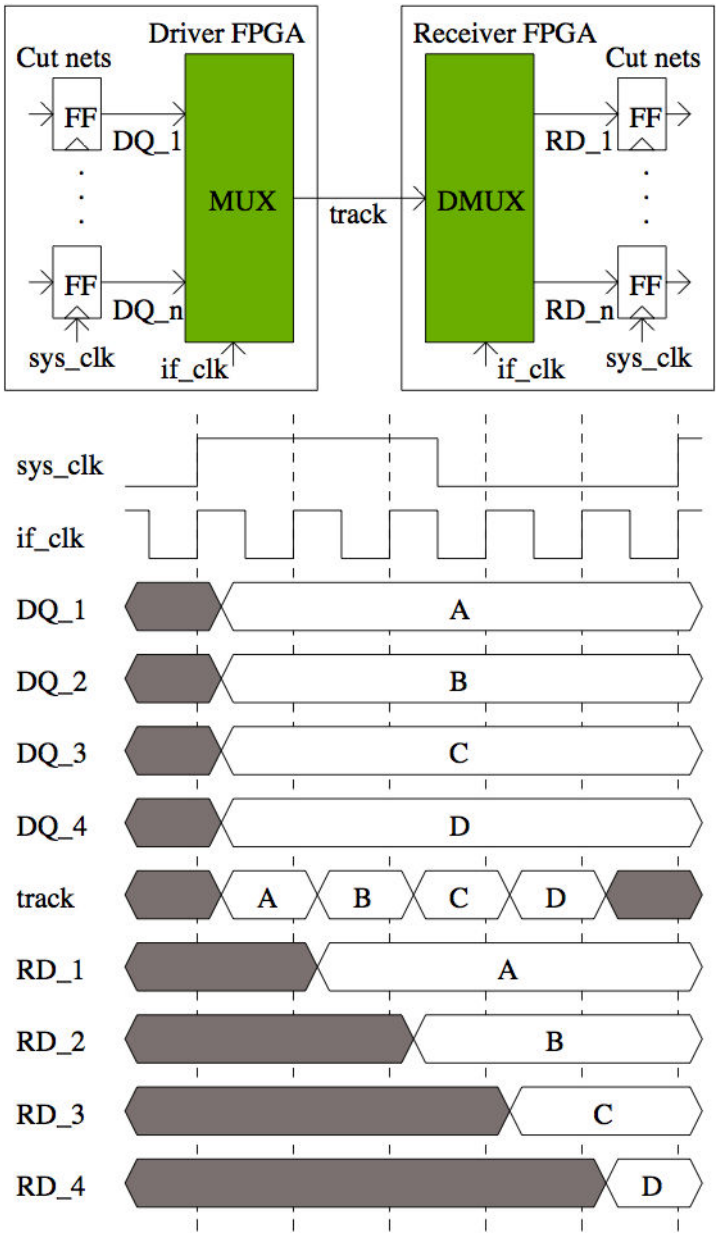


Figure 2.15: *The basic Time-Division-Multiplexing architecture*

There are three inter-FPGA communication architectures. Different architectures have different fast clock frequencies and different latencies, thus influence the performance.

- Logic Multiplexing: Logic blocks of the FPGA fabric are used to implement the multiplexing for inter-FPGA communication. The inter-FPGA data rate is ~ $125Mbps$, thus the system clock frequency is constrained. When the TDM ratio is 4, the system clock frequency is ~ $10MHz$ [Amos et al., 2011].

- ISERDES/OSERDES: Dedicated output parallel-to-serial converters (OSERDES) and input serial-to-parallel converters (ISERDES) are used for inter-FPGA communication. The data

rate is ~ $1Gbps$ [Amos et al., 2011]. This technique is widely used for multi-FPGA prototyping as it can achieve higher system clock frequency. When the TDM ratio is 4, the system clock frequency is ~ $30MHz$ [Pinmux, 2014].

- Multi-Gigabit Transceiver (MGT): Configurable hard-macros MGTs are implemented for inter-FPGA communication. The data rate can be as high as ~ $10Gbps$ [MGT, 2014]. Nevertheless, the MGT has a high latency (~ 30 fast clock cycles) that limits the system clock frequency and only a few is available. When the TDM ratio is 4, the system clock frequency is ~ $7MHz$ [Tang et al., 2014]. In addition, the communication between MGTs is not error-free. They come with a non-null bit error rate (BER). Therefore, at this moment, MGT is not used as inter-FPGA communication architecture in multi-FPGA prototyping.

2.3.2 Logic Multiplexing

In Logic Multiplexing, logic blocks of the FPGA fabric are used for inter-FPGA communication as shown in Figure 2.16. The inter-FPGA communication data rate is ~ $125Mbps$ [Amos et al., 2011]. The synchronous method for Logic Multiplexing is often system-synchronous [Amos et al., 2011] [SerialIO, 2014]. The system clocks for two FPGAs are generated by FPGA integrated PLL (Phase Locked-Loop) and have the same frequency with the clocks from the same Board Oscillator, thus are synchronous with each other. The fast clock is also generated by PLL in each FPGA, but is a frequency multiplier of the system clock. Therefore, the fast clock is synchronous with the system clock in each FPGA [Bilgic, 1982]. As the frequency of the fast clock is a frequency multiplier of the system clock, the fast clocks in two FPGAs are synchronous.

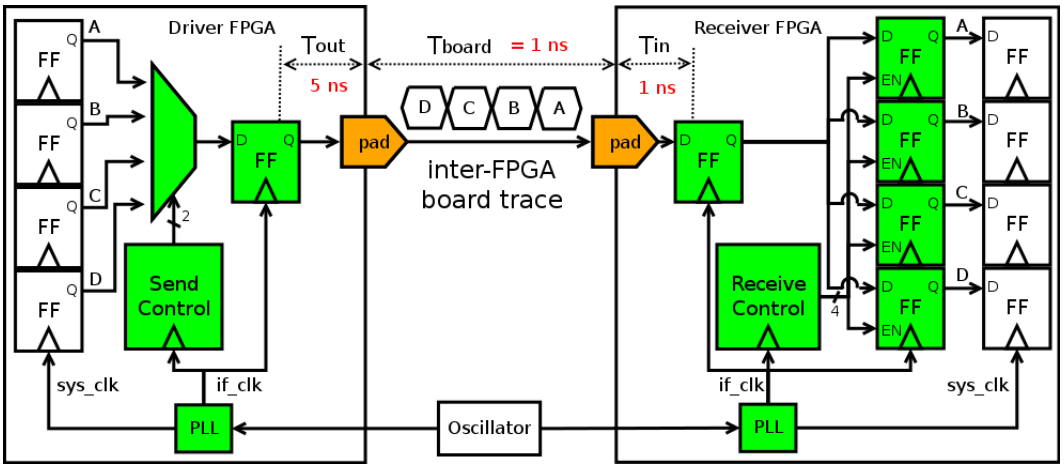


Figure 2.16: *Logic Multiplexing for Multi-FPGA Prototyping*

According to [Amos et al., 2011] [Pinmux, 2014], the critical path of multi-FPGA platforms is inter-FPGA tracks due to the output delay T_{out} (delay from flip-flop to pad, $5ns$), the inter-FPGA board trace delay T_{board} ($1ns$), the input delay T_{in} (delay from pad to flip-flop, $1ns$) and the tolerance delay $T_{tolerance}$ (safe margin for board clock distribution and etc., $1ns$). The total

delay T is the sum of all these delays as shown in Equation 2.1. Therefore, $T = 8ns$ and the fast clock frequency $if\_clk = \frac{1}{T} = 125MHz$ (thus, the data rate is $125Mbps$ due to the transmission is single data rate).

$$T = T_{out} + T_{board} + T_{in} + T_{tolerance} \tag{2.1}$$

The maximum number of cut nets passing through one logic multiplexer is noted as mux_logic. According to [Amos et al., 2011], the latency in Logic Multiplexing is $(mux\_logic + 3)$ fast clock cycles as there are 3 flip-flops in the datapath of the fast clock domain. Therefore, the relationship between the system clock frequency sys_clk and the fast clock frequency if_clk is shown in Equation 2.2. In Figure 2.16, if_clk is $125MHz$, mux_logic is 4 and sys_clk is $17.86MHz$.

$$sys\_clk = \frac{if\_clk}{lat} = \frac{125}{mux\_logic + 3}MHz \tag{2.2}$$

2.3.3 ISERDES/OSERDES

In ISERDES/OSERDES, dedicated output parallel-to-serial converters (OSERDES) and input serial-to-parallel converters (ISERDES) are used for inter-FPGA communication as shown in Figure 2.17. The Low-Voltage Differential Signaling (LVDS) is a signaling standard that provides high-speed data transfers by using a pair of FPGA I/O pins. Using ISERDES/OSERDES with LVDS, the inter-FPGA communication data rate for the targeted multi-FPGA board can be more than $1Gbps$ [Pinmux, 2014]. This architecture propagates the clock on a parallel path to the data for the synchronization and this synchronous method is source-synchronous [SerialIO, 2014]. The system clocks of two FPGAs have the same frequency due to that they are from the same Board Oscillator, thus are synchronous with each other. The fast clock of the source FPGA is generated by the PLL and is a frequency multiplier of the system clock, thus are synchronous with the system clock. Then, the fast clock is propagated from the source FPGA to the destination FPGA. Therefore, the fast clock of the destination FPGA is synchronous with that of the source FPGA. According to [SelectIO, 2014], a bank is a group of FPGA I/O pins that share a common resource such as one power supply or one output current reference. Several parallel ISERDES/OSERDES can be instantiated together with only propagating one clock, when the outputs of the source FPGA (resp. the inputs of the destination FPGA) are the FPGA I/O pins from the same bank. If not, one pair of pins per bank between FPGAs is reserved to propagate the clock instead of the user data due to FPGA technology limitation. The pair of FPGA I/Os that receives the clock need to be clock capable. This kind of inter-FPGA communication is called bank-to-bank communication.

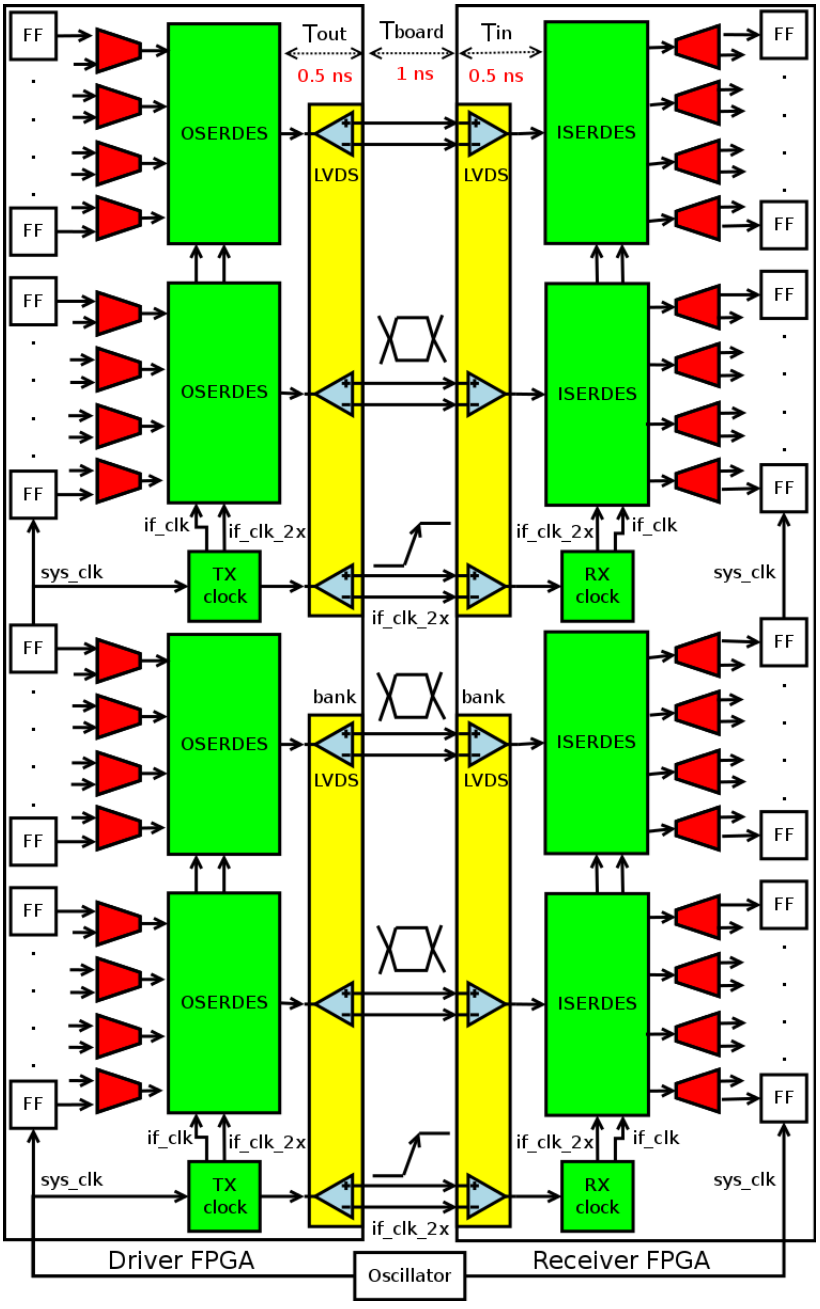


Figure 2.17: *ISERDES/OSERDES for Multi-FPGA Prototyping*

In the Xilinx FPGA Virtex-7, ISERDES/OSERDES can have programmable widths of 2, 3, 4, 5, 6, 7, or 8 bits [SelectIO, 2014]. The recommended width is 4 for optimal performance [Pinmux, 2014]. In this manuscript, a 4-bit wide SERDES is instantiated as depicted in Figure 2.17. If the number of cut nets is not a multiple of 4, some OSERSES inputs (resp. ISERDES outputs) will be left unconnected. If the number of cut nets is still exceeding the capacity of transmission after implementing the SERDES, w-bit wide logic Multiplexer/De-Multiplexer will be added. When there is no logic Multiplexer/De-Multiplexer implemented, w equals 1. The maximum number of cut nets passing through one ISERDES/OSERDES is noted as mux_serdes, therefore $w =$

$ceil(\frac{mux\_serdes}{4})$.

In [Pinmux, 2014], the system clock frequency for ISERDES/OSERDES is measured. The SERDES need another clock (noted as if_clk_2x), which is a frequency multiplier of the fast clock [SelectIO, 2014]. When the data rate is $1Gbps$, $if\_clk\_2x = 500MHz$ (due to that the transmission is double data rate) and the fast clock frequency is $250MHz$. The latency of ISERDES (resp. OSERDES) is 2 fast clock cycles [SelectIO, 2014] and $1 + w$ fast clock cycles are needed for the start pattern and w-bit wide logic Multiplexer/De-Multiplexer. The total delay T from OSERDES to ISERDES is the sum of the output delay T_{out} (delay from OSERDES to pad, $0.5ns$), the inter-FPGA board trace delay T_{board} ($1ns$) and the input delay T_{in} (from pad to ISERDES, $0.5ns$), thus is $2ns$. There is no need for the tolerance delay in ISERDES/OSERDES. Therefore, the total latency is ($(5 + w)$ fast clock cycles + $2ns$). As the data rate is $1Gbps$, the formula of the total latency is changed to be $\frac{(5+w)*1ns+2ns}{1(ns/fast\ clock\ cycle)} = (7+w)\ fast\ clock\ cycles$. Therefore, the relationship between the system clock frequency sys_clk and the fast clock frequency if_clk is shown in Equation 2.3. In Figure 2.17, if_clk is $250MHz$, if_clk_2x is $500MHz$, mux_serdes is 8 and sys_clk is $27.78MHz$.

$$sys\_clk = \frac{if\_clk}{lat} = \frac{250}{7 + ceil(\frac{mux\_serdes}{4})}MHz \tag{2.3}$$

2.3.4 Logic Multiplexing VS ISERDES/OSERDES

Logic Multiplexing and ISERDES/OSERDES are compared in the data rate, the complexity and the constraint as shown in Table 2.2. ISERDES/OSERDES achieves higher data rate than Logic Multiplexing with better usage of FPGA resources (i.e. dedicated blocks) to implement the serialization. The added IPs in ISERDES/OSERDES has a high complexity while only a simple multiplexer and some control logic is required in Logic Multiplexing. ISERDES/OSERDES is only usable on the platforms which take the bank-to-bank limitation into consideration, while Logic Multiplexing has no constraint.

Table 2.2: Logic Multiplexing VS ISERDES/OSERDES

	Logic Multiplexing	ISERDES/OSERDES
Data Rate	LOW (~ 125Mbps)	HIGH (~ 1Gbps)
Complexity	LOW	HIGH
Constraint	NO	bank-to-bank

The problem of complexity can be resolved by partitioning tools and keeping FPGA utilization low. Nevertheless, the bank-to-bank limitation inconveniences PCB designers. This limitation has already been taken into consideration in the off-the-shelf platform [DINI, 2014] and the cabling platform [HAPS, 2014] [Prodesign, 2014], which consist of ready-made boards. When designing a custom platform, if ISERDES/OSERDES is chosen as the inter-FPGA communication architecture, the inter-FPGA physical tracks need to be bank-to-bank.

Even though ISERDES/OSERDES has a higher data rate than Logic Multiplexing, one inter-FPGA track in ISERDES/OSERDES consumes a pair of FPGA I/Os in mode LVDS and several inter-FPGA tracks in ISERDES/OSERDES need to be used to propagate the clock. Therefore, the achieved performance of Logic Multiplexing and ISERDES/OSERDES need to be compared in different TDM ratios. The DNV7F2A board as shown in Figure 2.18, which is afforded by the company Dini Group [DINI, 2014], is used as the platform.

There are two Xilinx Virtex-7 2000T FPGAs, which have the package FHG1761 and the speed grade -1. The design is partitioned into two parts and implemented into two FPGAs. There are 300 inter-FPGA tracks between these two FPGAs. According to [Inagi et al., 2010], only the cut nets that have the same driver FPGA and the same receiver FPGAs, can be multiplexed together. In this manuscript, all the cut nets are from one FPGA (i.e. FPGA A) to another FPGA (i.e. FPGA B) in order to facilitate the comparison.

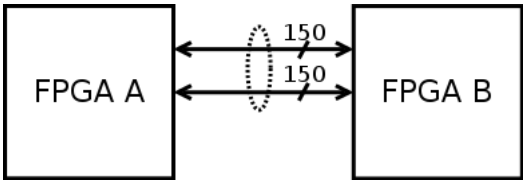


Figure 2.18: *The DNV7F2A board*

The relationship between the system clock and the fast clock of Logic Multiplexing (Equation 2.2) and ISERDES/OSERDES (Equation 2.3) is validated in the DNV7F2A board with the testbench LFSR [LFSR, 2014]. LFSR is a testbench that can reconfigure the number of cut nets. The result of the performance comparison is shown in Figure 2.19. The X-axis is the TDM ratio and the Y-axis is the system clock frequency. When the TDM ratio is 1, there is no need for multiplexing. The performance is compared when the TDM ratio is more than 1. If the TDM ratio is noted as $ratio$, there will be $300 * ratio$ cut nets due to that there are 300 inter-FPGA tracks between two FPGAs. When Logic Multiplexing is implemented, one inter-FPGA track is used to propagate one multiplexed data, the maximum number of cut nets passing through a logic multiplexer $mux\_logic = \frac{300*ratio}{300} = ratio$ and the system clock frequency sys_clk is calculated according to Equation 2.2. FPGA I/Os connected to inter-FPGA tracks in the DNV7F2A board are made bank-to-bank. Therefore ISERDES/OSERDES can be implemented in the DNV7F2A board (thus 150 pairs of inter-FPGA tracks in mode LVDS). In one FPGA, the I/Os connected to these 150 pairs of inter-FPGA tracks belong to 8 different banks. Therefore, 8 pairs of inter-FPGA tracks are reserved for propagating the clock and 142 pairs of inter-FPGA tracks are used for propagating the data. When ISERDES/OSERDES is implemented, the maximum number of cut nets passing through one ISERDES/OSERDES $mux\_serdes = \frac{300*ratio}{142}$ and the system clock frequency sys_clk is calculated according to Equation 2.3.

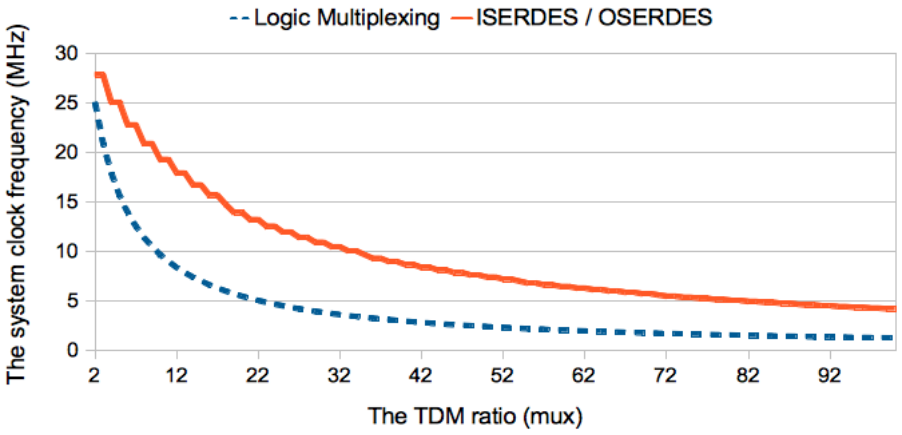


Figure 2.19: *Logic Multiplexing VS ISERDES/OSERDES*

The results show that as the TDM ratio is increasing, the achieved performance in Logic Multiplexing and ISERDES/OSERDES is reducing. In a duo-FPGA platform, ISERDES/OSERDES can always have higher performance than Logic Multiplexing. Nevertheless, the performance in ISERDES/OSERDES steps down instead of sliding down as in Logic Multiplexing, due to that the latency of SERDES is not changed when several OSERDES inputs (resp. ISERDES outputs) are not connected.

2.3.5 Conclusion of Inter-FPGA Communication Architectures

Multi-FPGA boards are widely used for rapid system prototyping. Nevertheless, these boards suffer from large timing delays in inter-FPGA communication compared to intra-FPGA net delays, as well as a limited bandwidth between FPGAs due to limited I/Os per FPGA. In order to solve the I/O pin limitation problem, cut nets are sent between FPGAs in a pipelined way using the Time-Division-Multiplexing technique. The maximum number of cut nets passing through one FPGA I/O is called the TDM ratio. There are three inter-FPGA communication architectures: Logic Multiplexing, ISERDES/OSERDES and Multi-Gigabit Transceiver (MGT). As we have seen, only Logic Multiplexing and ISERDES/OSERDES are today used for Time-Division-Multiplexing in the multi-FPGA based prototyping. In this Section, the achieved performances of these two multiplexing architectures are compared in different TDM ratios. Experiments are done in a duo-FPGA platform with the testbench LFSR to validate the achieved performance. The results show that as the TDM ratio is increasing, the multi-FPGA system performance is worsening and ISERDES/OSERDES can always have higher performance than Logic Multiplexing.

2.4 Conclusion

In this Chapter, the related work of the manuscript and the background of inter-FPGA communication are discussed.

Despite the fact that hardware/software integration is a more and more concerning problem, few papers have been published regarding multi-FPGA prototyping platforms. As there are three different multi-FPGA prototyping platforms, the conclusions of the related works are classified into four parts: First, the existing tools do not automatically route and multiplex cut nets in multi-point tracks, which waste FPGA I/Os that are already a scarce resource. In Chapter 3, which focuses on the off-the-shelf platform, we will target the direct architecture and we propose to spare FPGA I/Os by automatically routing and multiplexing multi-terminal nets over multi-point tracks. Second, the existing design flows for creating a custom platform are mainly manual and iterative (thus are time-consuming), and the optimization of the cost and the performance depends on the designers' experience. In Chapter 4, which focuses on the custom platform, we propose an automatic design flow for creating a custom multi-FPGA platform. Third, according to our knowledge, there is no tool to automatically have a solution for the cable distribution. In Chapter 5, which focuses on the cabling platform, we propose a cabling platform with an algorithm to automatically optimize the cable distribution and the external interface placement. Finally, different platforms are compared.

The performance of multi-FPGA platforms is limited by the inter-FPGA tracks. As there are fewer available inter-FPGA tracks than the number of cut nets, several cut nets need to be multiplexed and sent together onto a single track on the platform. There are two multiplexing techniques used for multi-FPGA based prototyping: Logic Multiplexing and ISERDES/OSERDES. The achieved performance of these two architectures is compared in an off-the-shelf duo-FPGA platform. The results show that as the TDM ratio is increasing, the multi-FPGA system performance is worsening and ISERDES/OSERDES can always have higher performance than Logic Multiplexing.

Chapter 3

Hardwired Off-the-Shelf Multi-FPGA Platform

3.1 Introduction

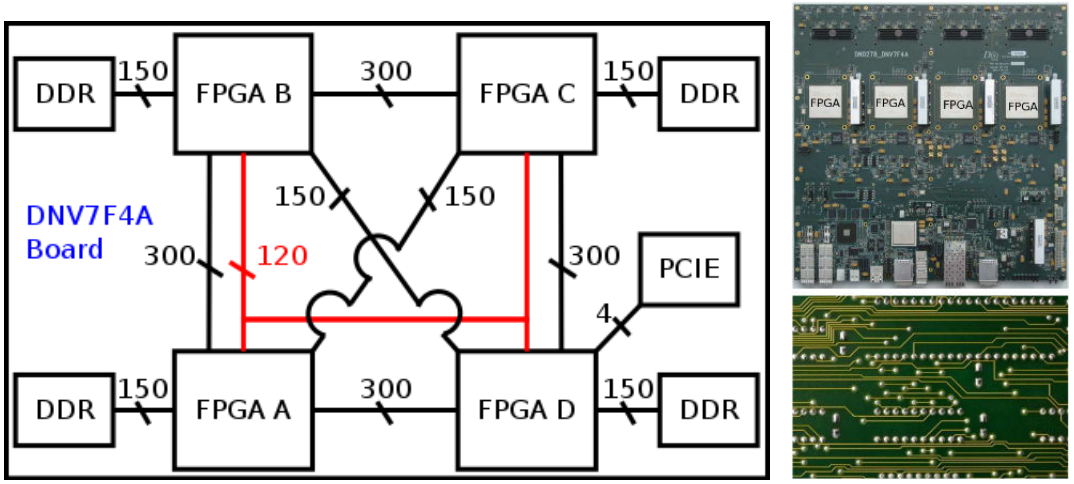
The previous Chapter has discussed the background (the state of the art of different multi-FPGA platforms and inter-FPGA communication architectures). The purpose of this Chapter is to discuss the off-the-shelf platform and present a typical design implementation flow for multi-FPGA platforms. The main contribution is to propose an algorithm that routes and multiplexes multi-terminal nets in multi-point tracks to spare FPGA I/Os, in order to improve the performance.

The rest of this Chapter is organized as follows. Section 3.2 is an overview of the off-the-shelf platform. Section 3.3 shows the typical implementation flow for multi-FPGA platforms. From Section 3.4 to Section 3.6, each step of the flow is detailed. In Section 3.7, experiments are conducted using Gaisler Research Benchmarks [Gaisler, 2014] and OpenCores Benchmarks [OpenCores, 2014]. Finally, Section 3.8 concludes this Chapter.

3.2 Platform Overview

In this manuscript, the off-the-shelf platform consists of a ready-made generic multi-FPGA board, where all the inter-FPGA connections are fixed and realized using PCB traces. The connections to external interfaces are also fixed but can be realized using PCB traces or connectors (connected to daughter boards). An example of such platform is the commercial platform DNV7F4A as shown in Figure 3.1, which is the latest platform from Dini Group [DINI, 2014]. The platform uses four FPGAs of the latest Xilinx FPGA (family: Virtex-7, device: 2000T, and package: FLG1925, thus FPGA type: XC7V2000TFLG1925). Each FPGA is connected with a DDR by connectors. Only the FPGA D is connected with a PCIE by connectors. Nevertheless, the connectors are fixed and specific. This means that the connectors to one type of the external interface (i.e. DDR) can not be transferred to the connectors to another type of the external interface (i.e. PCIE).

If the prototyped design has less than 4 DDRs, several FPGA I/Os connected with DDRs are wasted. If the prototyped design has a PCIE, the PCIE part of the prototyped design is forced to be placed in FPGA D. These constraints add to the complexity of the design partitioning and may reduce the performance. The number of inter-FPGA tracks is shown in Figure 3.1(a). All the inter-FPGA tracks are fixed, bank-to-bank and LVDS compliant. There are two types of inter-FPGA tracks in the multi-FPGA prototyping platform: 2-point tracks and multi-point tracks. The number of multi-point tracks in the example is 120. The number of 2-point tracks between a pair of FPGAs are generic and balanced: 300 connections in all the horizontal and vertical pairs, and 150 connections in all the diagonal pairs. When implementing a specific design, these generic and balanced connections constraint the achieved performance. Figure 3.1(b) shows the picture of the platform and the PCB traces.



(a) The architecture (b) The picture

Figure 3.1: *The off-the-shelf multi-FPGA platform*

3.3 Implementation Flow of Multi-FPGA Platforms

Implementing multi-million gates SoCs into multi-FPGA platforms is very challenging. An automatic typical implementation flow is shown in Figure 3.2. The typical implementation flow, which starts by inputting the design RTL, can be divided in three main steps:

1. Logic Synthesis: The input design is synthesized targeting FPGAs (from RTL to design netlist).

2. Design Partitioning: As the SoC/ASIC design is bigger than the FPGA, the design netlist is partitioned into several parts according to the number of FPGAs in the platform. Each part's capacity fits in a single FPGA. The signals crossing design's parts located in different FPGAs are called cut nets.

3. Design Routing / Multiplexing: The cut nets are routed meaning that a cut net is allocated to an inter-FPGA track in the platform. As there are fewer available inter-FPGA tracks than the number of cut nets, several cut nets need to be multiplexed and sent together onto a single track on the platform.

The typical implementation flow outputs the design sub-netlists and multiplexing IPs are added if needed. Finally, these design sub-netlists run respectively through the FPGA PnR tools (i.e. ISE Design Suite [ISE, 2014]) to generate the bitstreams for each FPGA, and the generated bitstreams are downloaded into the platform to model the design.

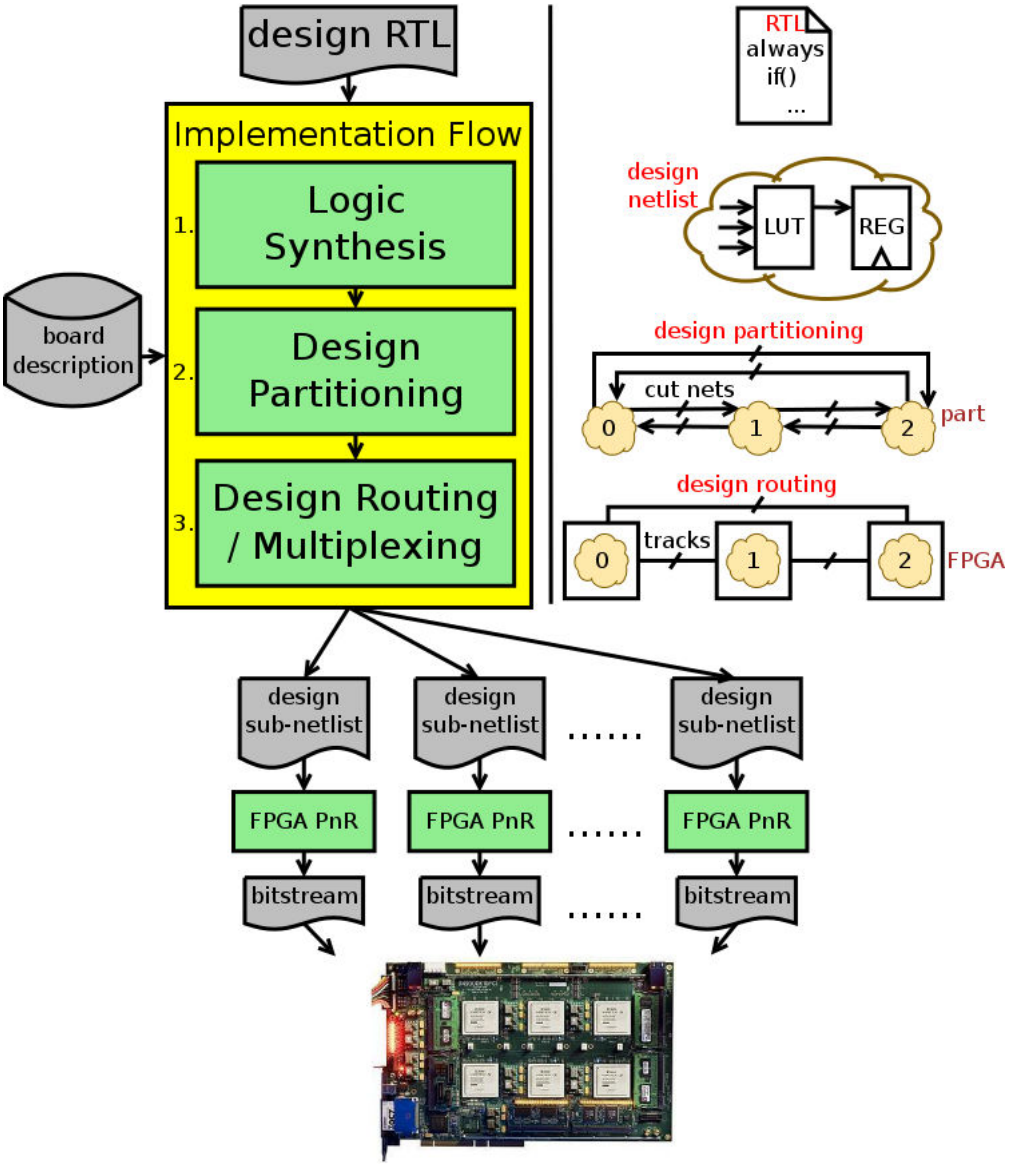


Figure 3.2: *Typical implementation flow for multi-FPGA platforms*

3.4 Logic Synthesis

The first step of the typical implementation flow is to synthesize the input design targeting FPGAs, which translates the design from RTL to netlist. After the logic synthesis, the logic capacity of the input design is represented by the number of logic elements contained. There are four kinds of logic elements which are taken into consideration: Look-Up Table (noted as LUT), Register (REG), Block RAM (RAM), and Embedded DSP (DSP). This step can be realized by Xilinx XST Synthesis tool [XST, 2014], Altera Quartus Synthesis tool [Quartus, 2014], or third party tools such as Synopsys Synplify tool [Synplify, 2014] and Mentor Graphics Precision tool [Precision, 2014].

3.5 Design Partitioning

Due to that the silicon area overhead of FPGA versus ASIC technology has been measured to be about 40x [Kuon and Rose, 2010], FPGA technology requires that an ASIC logic design should be partitioned across multiple FPGA devices to achieve the necessary logic capacity. In the partitioning, the design is split into pieces small enough to fit into the individual FPGAs.

In the following of this Section, we will introduce the notion of design partitioning, which is not the objective of this manuscript but only gives an overview. The partitioning has been an area of active research from 1970s [Kernighan and Lin, 1970] to 2000s [Selvakkumaran et al., 2004]. There have been numerous approaches tried, and several promising approaches have emerged. While many of these have been focused on bi-partitioning, or breaking into exactly two partitions [Kernighan and Lin, 1970] [Fiduccia and Mattheyeses, 1982] [Krishnamurthy, 1984] [Hagen and Kahng, 1992] [Dutt, 1993] [Bui and Moon, 1994] [Riess et al., 1994] [Yang and Wong, 1994], there has been work on extending them to k-way partitioning [Sanchis, 1989] [Chan et al., 1994] [Woo and Kim, 1993] that splits into more than two partitions.

To obtain more than two partitions, there are two major approaches:

- Recursive bi-partitioning (also called multi-level bi-partitioning): Partitions are split recursively until the desired number of partitions is obtained.
- Simultaneous computation of all partitions (also called k-way partitioning).

In the study of [Yarack and Carletta, 2000], several move-based k-way approaches have been compared. All these approaches have in common that for k-way partitioning the amount of memory needed grows quadratic with k. Therefore, k-way approaches can not be applied, if k becomes large. Then, the approaches are restricted on recursive bi-partitioning.

There are two kinds of constraints in the partitioning: connection constraints (the distribution of inter-FPGA connections) and size constraints (the number of logic elements in one FPGA). With these constraints, the problem of optimally partitioning a netlist (modeled as hypergraph) is

known to be NP-hard [Garey and Johnson, 1990]. And the netlist partitioning algorithms (or called as the hypergraph partitioning algorithms) are heuristics.

However, previous presented bi-partitioning works have primarily focused on problems where there are no restrictions on how the partitions are interconnected (that is, there is no reason to prefer or avoid connections between any pair of partitions). Unfortunately, in many multi-FPGA platforms only a subset of the FPGAs are connected, and routing between FPGAs not directly connected (or less directly connected) will use many more resources than routing between connected (or highly connected) FPGAs. Works that have handled the limited connectivity problem take a significant amount of time [Roy and Sechen, 1993], possibly even exponential in the number of partitions [Vijayan, 1990].

[Kuznar et al., 1993] proposed an algorithm to partition a given netlist into multiple FPGA types to minimize total cost, where each FPGA type in a given library can have distinct price, size, and pin capacity. Their method recursively applies a variant of [Fiduccia and Mattheyeses, 1982] bi-partitioning which allows some uphill moves. In subsequent work, [Kuznar et al., 1994] allow logic resources (logic elements) to be duplicated, i.e., they introduce functional replication to minimize the total cost of FPGAs and the total number of cut nets. Finally, [Chou et al., 1994] [Huang and Kahng, 1995] have proposed an algorithm to partition a circuit into instances of a single FPGA type, such that the number of FPGAs is minimized. Their algorithms significantly improve over recursive [Fiduccia and Mattheyeses, 1982] on benchmarks.

The approach of [Hauck and Borriello, 1995] to the multi-FPGA partitioning problem is to harness the work on standard bi-partitioning for some restricted situations (i.e. connection constraints and size constraints on each partition). This is done by recursively applying the bi-partitioning algorithms to the circuit until it is cut into the required number of pieces. Then, the multi-level bi-partitioning algorithm has been improved to dedicate to larger number of partitions in [Alpert et al., 1997] [Karypis et al., 1997] [Drechsler et al., 2002].

However, modern FPGA architectures incorporate heterogeneous logic resources (e.g., LUT, REG, RAM, DSP and etc.). Thus, the partitioning algorithm must now ensure that the logic resources used in each partition can be accommodated by the logic resources provided at the different regions of the FPGA. For example, a partitioning solution that places most of the REG on one side of the bi-partitioning and most of the RAM on the other side of the bi-partitioning, even if it is balanced in terms of the total number of logic resources on either side of the cut, it is not very useful for FPGA placement as it may over-subscribe these two logic resource types.

As a result, previous presented partitioning algorithms [Kuznar et al., 1994] [Chou et al., 1994] [Huang and Kahng, 1995] [Hauck and Borriello, 1995] [Alpert et al., 1997] [Cong et al., 1997] [Wichlund and Aas, 1998] [Karypis et al., 1997] [Cong and Lim, 2000] can not be used to develop partitioning-based placement methods for FPGAs with heterogeneous logic resources, as they can lead to partitions that have highly unbalanced logic resource requirements. To illustrate this, a multi-level hypergraph partitioning algorithm is used (hMetis [Karypis and Kumar, 1998]) to bi-partition twelve different circuits synthesized for the Xilinx Virtex-2 architecture, which

contain cells that map to different logic resources. In [Selvakkumaran et al., 2004], a new class of multi-resource hypergraph bi-partitioning algorithms are presented to simultaneously balance the different logic resources assigned to each one of the partitions.

Nowadays, there are four commercial tools: Cadence Protium tool [Protium, 2014], Synopsys Certify tool [Synopsys, 2014], Auspy tool [Auspy, 2014] and Flexras Wasga tool [Flexras, 2014], targeting at the partitioning problem. In this manuscript, the design partitioning is done by the Flexras Wasga tool provided by the company Flexras Technologies [Flexras, 2014]. The partitioning tool integrates the multi-resource multi-level hypergraph partitioning algorithm taking the connection constraints and the size constraints into consideration. The objective of the design partitioning is to minimize the total number of cut nets, which contributes to increase the performance of multi-FPGA prototyping.

3.6 Design Routing

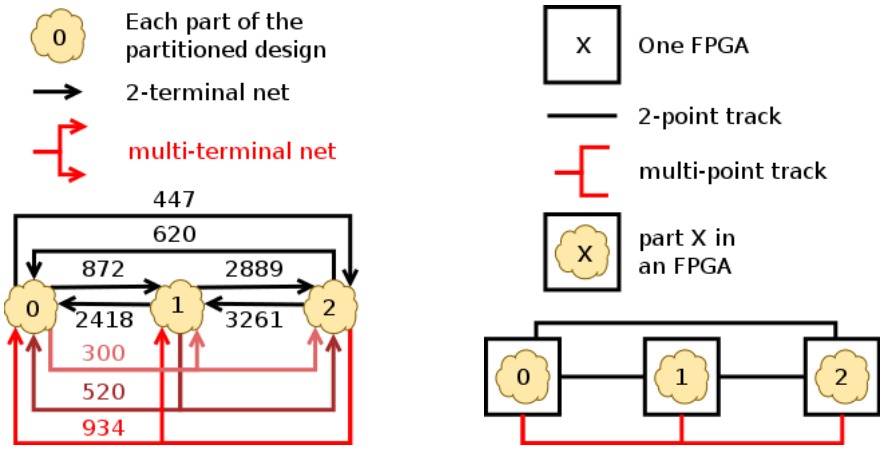
After the partitioning, the design has been divided into several parts. Then, the partitioned design will be routed in a multi-FPGA platform. In the partitioned design, there are two types of cut nets as shown in the example of Figure 3.3(a). One type of cut nets has one driver and one receiver (black arrow from one driver to one receiver), called 2-terminal nets. The second type of cut nets has one driver and multiple receivers (colored arrows from one driver to multiple receivers), called multi-terminal nets. Off-the-shelf multi-FPGA prototyping platforms come with two types of inter-FPGA tracks as shown in Figure 3.3(b). One type of tracks connects only two FPGAs (black line), called 2-point tracks. Another type of tracks connects more than two FPGAs (red line), called multi-point tracks. In Section 2.2.1.2, an iterative congestion-aware routing algorithm is proposed by [Turki et al., 2013]. Nevertheless, this algorithm only uses 2-point tracks and routes both 2-terminal and multi-terminal nets through a sequence of 2-point tracks. In this manuscript, we enhance this algorithm to efficiently route multi-terminal nets in multi-point tracks in order to save FPGA I/Os.

3.6.1 Multi-Terminal Net Routing

In the manuscript, a multi-point track is assumed to connect all the FPGAs in a multi-FPGA board. Therefore, if there are m FPGAs in the board, multi-point tracks are m-point. The number of terminals for a cut net in the design, which is noted as n, varies from 2 to m. An example of routing a multi-terminal net in 2- and multi-point tracks will be studied in the following.

3.6.1.1 Multi-terminal net routing in 2-point tracks

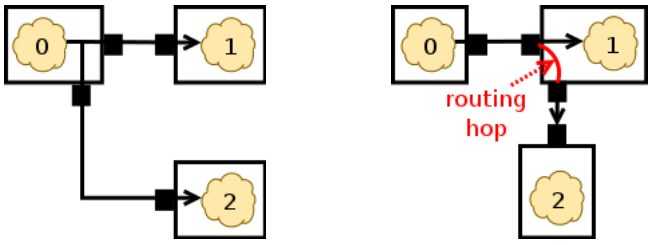
Routing a multi-terminal net in 2-point tracks is shown in Figure 3.4. The driver is in FPGA 0, one receiver is in FPGA 1 and another receiver is in FPGA 2. There are two ways to route this cut net. 4 FPGA I/Os are used in both cases.



(a) The partitioned design (b) The multi-FPGA platform

Figure 3.3: *The partitioned design and the corresponding multi-FPGA platform*

- If there are tracks between FPGA 0 and FPGA 1 and between FPGA 0 and FPGA 2, the multi-terminal net will be routed as shown in Figure 3.4(a).
- If there is no track between FPGA 0 and FPGA 2, but there are tracks between FPGA 0 and FPGA 1 and between FPGA 1 and FPGA 2, the multi-terminal net will be routed as shown in Figure 3.4(b). Nevertheless, this results a routing hop (connecting 2 FPGA I/Os without passing the logic of the design) due to that an intermediate FPGA is needed from the Driver in FPGA 0 to the Receiver in FPGA 2. The routing hop decreases the performance which will be detailed in Section 3.6.3.



(a) without routing hop (b) with routing hop

Figure 3.4: *Routing a multi-terminal net in 2-point tracks*

Note that, for a n-terminal net, routing in 2-point tracks needs $2(n-1)$ FPGA I/Os.

3.6.1.2 Multi-terminal net routing in multi-point tracks

Routing a multi-terminal net in a multi-point track is shown in Figure 3.5. This multi-point routing needs 3 FPGA I/Os and results no routing hop. Therefore, the advantages of routing a multi-terminal net in a multi-point track instead of 2-point tracks are to reduce the used FPGA I/Os and avoid routing hops, and consequently increase the performance.

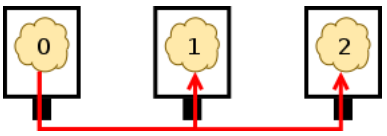


Figure 3.5: *Routing a multi-terminal net in a multi-point track*

Note that, for a n-terminal net, routing in a m-point track ($m \geq n$) needs m FPGA I/Os.

Nevertheless, if m is much more than n, precisely if m is more than $2(n-1)$, using the multi-point track can be a waste of FPGA I/Os.

3.6.2 Routing Algorithm

The proposed routing algorithm is shown in Figure 3.6. The pink areas are the added functions. The algorithm tries to minimize the maximum number of cut nets passing through one multiplexer (noted as mux). If Logic Multiplexing is chosen as the inter-FPGA communication architecture, $mux = mux\_logic$. If ISERDES/OSERDES is chosen, $mux = mux\_serdes$.

3.6.2.1 Compute initial mux

After loading the design and the board model, the algorithm will calculate initial mux by obtaining the maximum ratio of cut nets and inter-FPGA tracks between each pair of FPGAs.

3.6.2.2 Group cut nets to GNet

The cut nets, which have the same driver FPGA and the same receiver FPGAs, are grouped together. The group, which is called GNet, will be put in one track. Each GNet contains a maximum of mux cut nets.

3.6.2.3 Enable routing in multi-point tracks

An example of a board with both 2- and multi-point tracks is shown in Figure 3.7(a). There are a 2-point track between FPGA 1 and FPGA 2 and a multi-point track connecting all the FPGAs.

- The existing routing algorithm [Turki et al., 2013] does not enable routing in multi-point tracks. The algorithm considers a multi-point track as a 2-point track. Executing the algorithm leads to the solution presented in Figure 3.7(b). The driver is in FPGA 0, one receiver is in FPGA 1 and another receiver is in FPGA 2. The algorithm considers the multi-point track as the 2-point track between FPGA 0 and FPGA 1. Therefore, in order to route from the driver in FPGA 0 to the receiver in FPGA 2, 4 FPGA I/Os and a routing hop are needed.

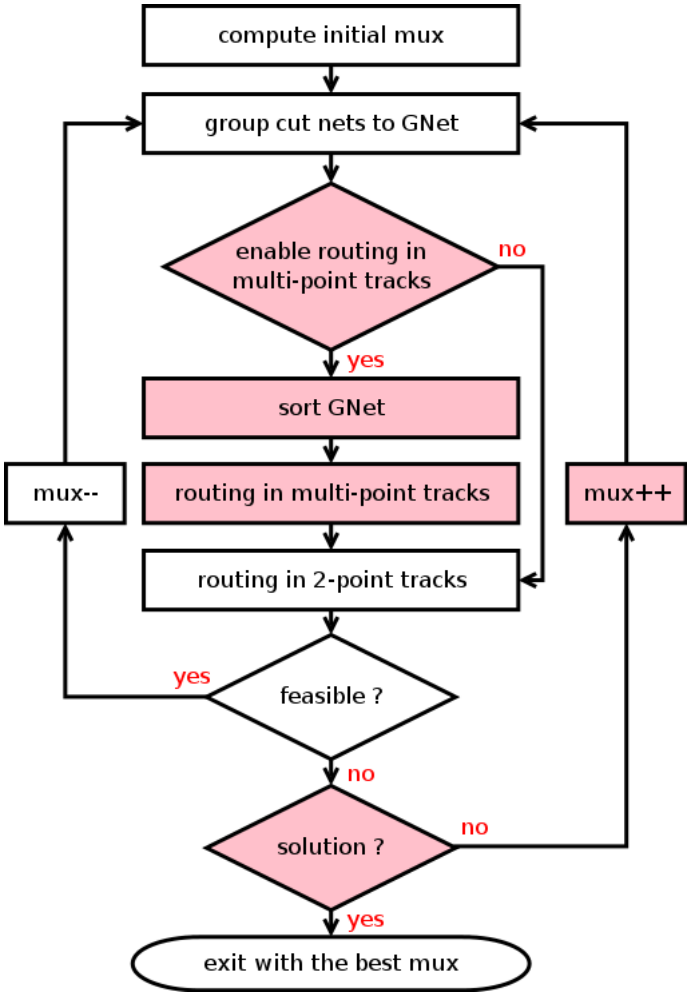
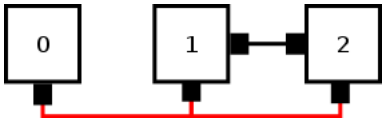


Figure 3.6: *The proposed routing algorithm*

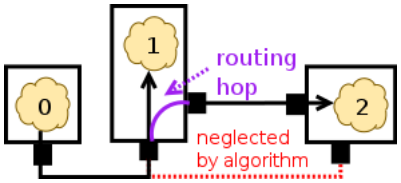
- The proposed routing algorithm enables routing in multi-point tracks, and leads to the solution shown in Figure 3.7(c). Therefore, in order to route from the driver to the receivers, 3 FPGA I/Os and no routing hop are needed.

3.6.2.4 Sort GNets and Routing in multi-point tracks

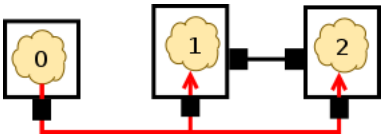
When the routing in multi-point tracks is enabled, the algorithm counts the number of multi-point tracks (noted as num_multi). Experiments have been done in an off-the-shelf multi-FPGA platform DN9000K10PCI [DINI, 2014] to verify the functionality of multi-point tracks, with the testbench LFSR [LFSR, 2014]. The results show that multi-point tracks are more critical than 2-point tracks in terms of performance. Therefore, the number of cut nets on a multi-point track should be a half of that on a 2-point track, in order to avoid that the critical path is the multi-point track and increase the performance. Thus, $\frac{num\_multi}{2}$ GNets, which have the largest number of terminals, are chosen and assigned on multi-point tracks. Each GNet is divided into 2 sub GNets



(a) Board (2- & multi-point tracks)



(b) Existing routing algorithm



(c) Proposed routing algorithm

Figure 3.7: *Routing a multi-terminal net in a multi-FPGA board*

and each sub GNet is assigned on one multi-point track.

3.6.2.5 Routing in 2-point tracks

PathFinder [McMurchie and Ebeling, 1995] was used primarily for routing intra-FPGA nets. It is adapted to deal with GNets and routes GNets in 2-point tracks. An iterative negotiation-based approach is used in PathFinder. During the first routing iteration, GNets are freely routed without paying attention to track sharing. Individual GNets are routed using Dijkstra's shortest path algorithm [Cormen et al., 2009]. At the end of the first iteration, tracks may be congested because multiple GNets try to use one track. During subsequent iterations, the cost of using a track is increased, based on the number of GNets that share the track, and the history of congestion on that track. Thus, GNets are forced to negotiate for tracks. If a track is highly congested, GNets which can use lower congestion alternatives are forced to do so. On the other hand, if the alternatives are more congested than the track, then a GNet may still use that track. Due to that one track can only contain one GNet, if there are two GNets in one track in the result of PathFinder, there is a conflict. A routing result is feasible only when the number of conflicts is 0.

An example is shown in Figure 3.8. There are two GNets: one GNet N1 from FPGA 0 (driver) to FPGA 3 (receiver) and another GNet N2 from FPGA 1 (driver) to FPGA 3 (receiver). There is one inter-FPGA track between FPGA 0 and FPGA 1, FPGA 0 and FPGA 2, FPGA 1 and FPGA 3, FPGA 2 and FPGA 3. N1 will be routed before N2. In the first iteration, there are two choices of paths for N1 according to Dijkstra's shortest path algorithm. One path is (FPGA 0 - FPGA 1 - FPGA 3). Another path is (FPGA 0 - FPGA 2 - FPGA 3). N1 is freely routed. In the example, N1 chooses the path (FPGA 0 - FPGA 1 - FPGA 3). The congestion (noted as C) of the tracks (between FPGA 0 and FPGA 1, FPGA 1 and FPGA 3) plus 1. N2 chooses its shortest path (FPGA 1 - FPGA 3). The congestion of the track (between FPGA 1 and FPGA 3) plus 1. Both N1 and N2 pass through the track between FPGA 1 and FPGA 3. Therefore, there is a conflict. In the second iteration, for N1, the path (FPGA 0 - FPGA 1 - FPGA 3) has higher congestion value than the path (FPGA 0 - FPGA 2 - FPGA 3). Therefore, N1 chooses the path (FPGA 0 - FPGA 2 - FPGA 3).

Then, N2 chooses its shortest path (FPGA 1 - FPGA 3). After that, a feasible solution has been found. The quality of the routing is independent from the routing order of GNets.

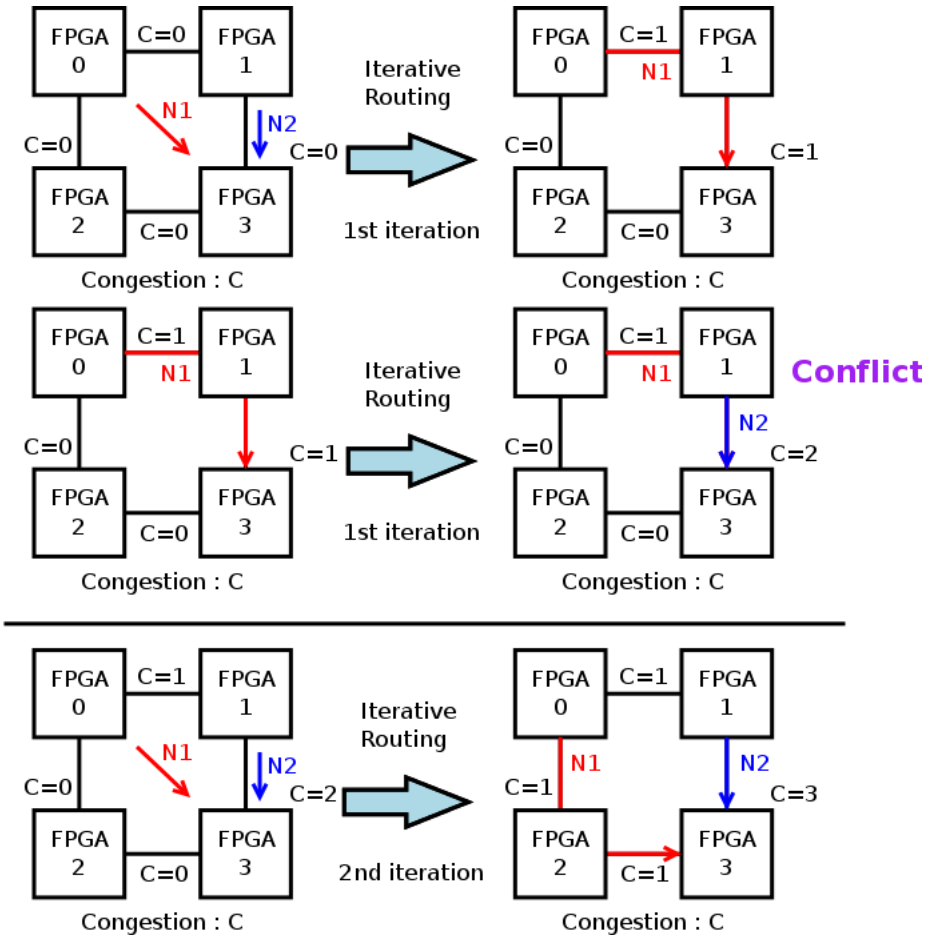


Figure 3.8: *Routing in 2-point tracks*

3.6.2.6 Minimize mux

The proposed algorithm tries to minimize mux as shown in Figure 3.6,

- If feasible, decrease the mux and iterate the previous steps beginning from grouping cut nets with the new mux.
- If not feasible, examine whether there is a feasible solution found before:
  - If so, exit with the mux of the previous feasible solution.
  - If not, increase the mux and iterate the previous steps with the new mux.

3.6.2.7 Results of the proposed routing algorithm

As discussed above, intermediate FPGAs may be necessary in routing 2-point tracks if there is no available track between the FPGA (the Driver) and the FPGA (the Receiver). Routing hop (noted as hop) is defined as the number of intermediate FPGAs needed. For example, the routing of GNet N1 in Figure 3.8 causes a routing hop. When the routing hop exists, the performance degrades. At the end of the proposed routing algorithm, the results consist of a group of feasible solutions (mux, hop). The optimal (mux, hop) will be chosen to achieve the highest performance.

3.6.3 Performance Evaluation

The inter-FPGA communication architectures have been presented in Section 2.3 and the achieved performance has been evaluated. Nevertheless, the routing hops are not taken into consideration. In this Section, the inter-FPGA communication architectures with routing hops will be proposed and the performance will be evaluated. As there are two inter-FPGA communication architectures (Logic Multiplexing and ISERDES/OSERDES), this Section is classified into two parts.

3.6.3.1 Logic Multiplexing

When there are one more intermediate FPGAs, the routing hops of Logic Multiplexing are realized in a pipelined way. There are one more flip-flops added in the datapath of the fast clock domain as shown in Figure 3.9, thus one more fast clock cycles in the latency. When considering routing hops, the achieved performance (in terms of the system clock frequency) calculated in Equation 2.2 will be modified as in Equation 3.1.

$$sys\_clk = \frac{if\_clk}{lat} = \frac{125}{mux\_logic + hop + 3} MHz \tag{3.1}$$

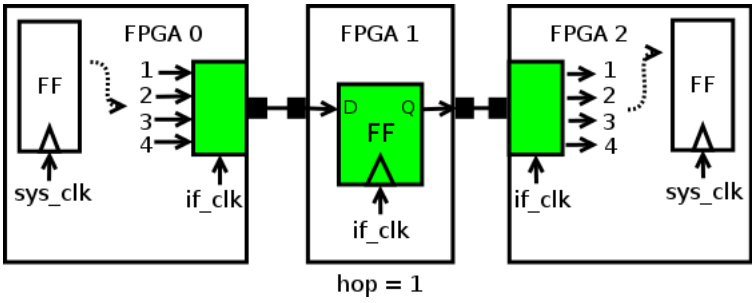


Figure 3.9: *routing hops in Logic Multiplexing*

3.6.3.2 ISERDES/OSERDES

When there are one more intermediate FPGAs, one more ISERDES/OSERDES pairs are added in the datapath as shown in Figure 3.10. In the intermediate FPGA (FPGA 1), the fast clock frequency

if_clk_0 in ISERDES is generated and propagated from FPGA 0, but the fast clock frequency if_clk_1 in OSERDES is generated inside FPGA 1. Therefore, there is a phase shift between if_clk_0 and if_clk_1, and there is a difficulty to realize the routing hops of ISERDES/OSERDES in a pipelined way. In this manuscript, data starts to propagate from ISERDES of FPGA 1 to OSERDES of FPGA 1 only when all the data is received in ISERDES of FPGA 1. Thus, when considering routing hops, the achieved performance calculated in Equation 2.3 will be modified as in Equation 3.2.

$$sys\_clk = \frac{if\_clk}{lat} = \frac{250\ MHz}{(7 + ceil(\frac{mux\_serdes}{4})) * (1 + hop)} \tag{3.2}$$

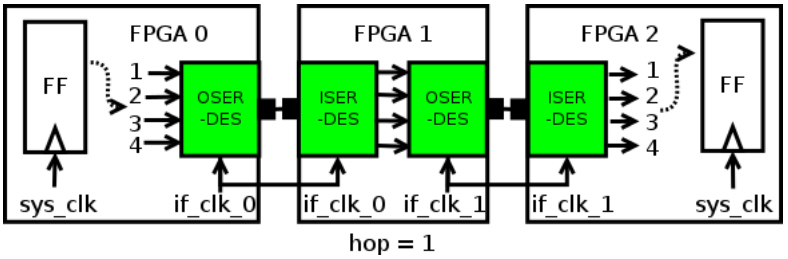


Figure 3.10: *routing hops in ISERDES/OSERDES*

In the routing process, different feasible pairs (mux, hop) can be generated. Different (mux, hop) pairs have different system clock frequencies according to Equation 3.1 and Equation 3.2. In the performance evaluation, the optimal (mux, hop) pair will be chosen to achieve the higher performance according to the chosen inter-FPGA communication architecture.

3.7 Experiments

3.7.1 Targeted Platform

The proposed routing algorithm is independent of FPGA types. The off-the-shelf multi-FPGA platform DN9000K10PCI [DINI, 2014] with both 2- and multi-point tracks is available and used for experiments. The architecture is shown in Figure 3.11. There are six identical FPGAs. They are Xilinx FPGAs [Virtex-5, 2014] with the family Virtex-5, device LX330, and the package FF1760 (noted as FPGA type: XC5VLX330FF1760). 1244160 Look-Up Table, 1244160 Register, 1152 Block RAM and 768 Embedded DSP are contained in this board. The number of tracks for each line is shown in the figure.

3.7.2 Results

In the experiments, four testbenches (leon2, leon3_avnet, leon3mp, netcard) from Gaisler Research Benchmarks [Gaisler, 2014] and three testbenches (vga_lcd, des_perf, ethernet) from Open-

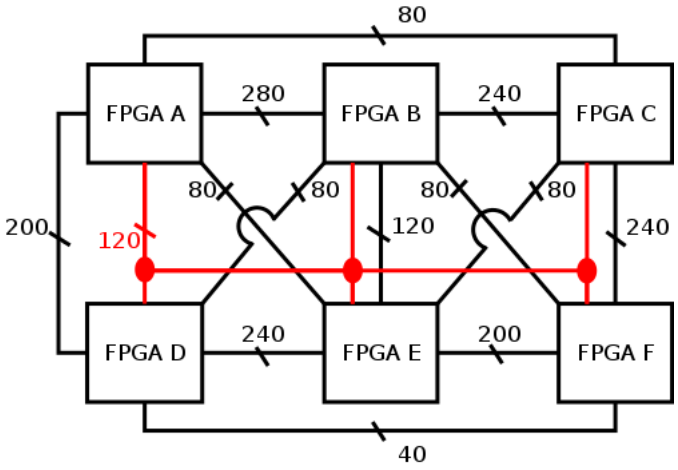


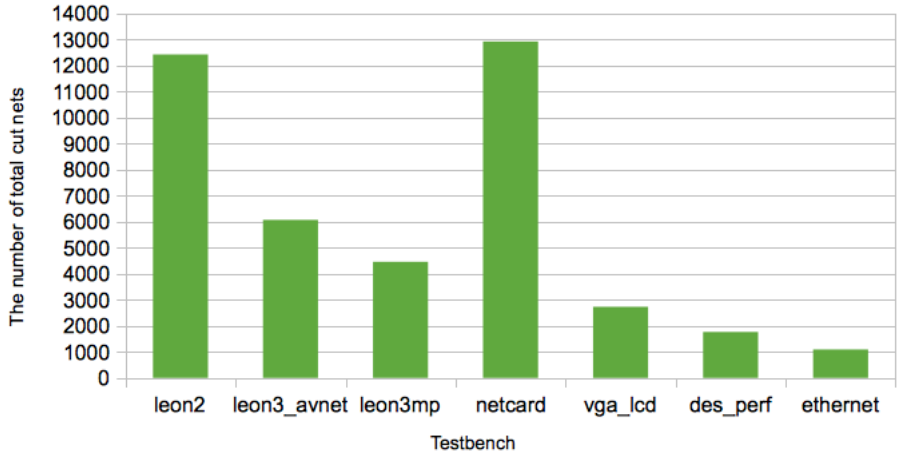
Figure 3.11: *The architecture of DN9000K10PCI*

Cores Benchmarks [OpenCores, 2014] will be used. Due to that the targeted off-the-shelf platform has 6 FPGAs ($m = 6$), these testbenches are partitioned into 6 parts with the commercial tool afforded by the company Flexras Technologies [Flexras, 2014] as shown in Table 3.1. Total cut nets for each testbench are shown and classified according to their number of terminals (noted as ter). The percentage of each ter for each testbench is shown in the Table.

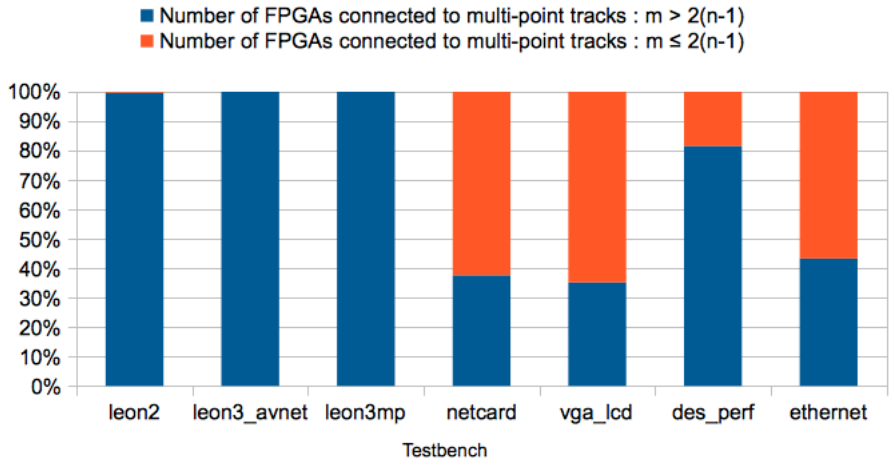
Table 3.1: Partitioning results of testbenches

testbench	Total Cut Nets	$m > 2(n-1)$		$m \leq 2(n-1)$		
		2-ter (%)	3-ter (%)	4-ter (%)	5-ter (%)	6-ter (%)
leon2	12430	42.84	56.85	0.31	0	0
leon3_avnet	6072	98.39	1.61	0	0	0
leon3mp	4460	98.57	1.41	0.02	0	0
netcard	12929	27.34	10.3	20.53	41.51	0.32
vga_lcd	2731	22.85	12.38	19.22	19.74	25.81
des_perf	1764	56.29	25.34	12.87	4.76	0.74
ethernet	1090	27.15	16.15	47.61	9	0.09

The number of total cut nets of most testbenches is largely more than the available FPGA I/Os (max. 1200) as shown in Figure 3.12(a). According to the previous discussion, routing an n-terminal net in a multi-point track can reduce the used FPGA I/Os when $m \leq 2(n-1)$ and can waste the FPGA I/Os when $m > 2(n-1)$ compared to routing in a 2-point track. In the testbenches (netcard, vga_lcd, des_perf), there are more than 18% of cut nets can benefit from multi-point routing as shown in Figure 3.12(b).



(a) the number of total cut nets



(b) the percentage of multi-terminal nets that can benefit from multi-point routing

Figure 3.12: *The partitioning results*

After the design partitioning, each testbench will be routed in the targeted off-the-shelf platform DN9000K10PCI. The results are shown in Table 3.2. The number of cut nets passing through one multiplexer is noted as mux ($mux = mux\_logic$ in case of Logic Multiplexing, $mux = mux\_serdes$ in case of ISERDES/OSERDES). The maximum number of intermediate FPGAs needed from the driving FPGA to the receiving FPGA (routing hops) is noted as hop. There are four scenarios depending on the global routing algorithm and the inter-FPGA communication architecture.

- The Turki's algorithm [Turki et al., 2013], Logic Multiplexing (noted as LM)
- The Turki's algorithm, ISERDES/OSERDES (noted as SERDES)
- The proposed algorithm, Logic Multiplexing
- The proposed algorithm, ISERDES/OSERDES

Table 3.2: The comparison of routing results with different algorithms

Inter-FPGA Communi-cation	testbench	Turki's algorithm			proposed algorithm			gain vs 2-point	gain vs LM
		mux	hop	sys_clk (MHz)	mux	hop	sys_clk (MHz)		
LM	leon2	19	2	5.21	18	2	5.43	4%	
	leon3_avnet	8	1	10.42	8	1	10.42	0	
	leon3mp	5	1	13.89	5	1	13.89	0	
	netcard	28	3	3.68	24	3	4.17	13%	
	vga_lcd	6	3	10.42	5	3	11.36	9%	
	des_perf	4	2	13.89	4	1	15.63	12%	
	ethernet	6	1	12.5	4	1	15.63	25%	
SERDES	leon2	68	0	10.42	68	0	10.42	0	92%
	leon3_avnet	32	0	16.67	32	0	16.67	0	60%
	leon3mp	18	0	20.83	18	0	20.83	0	50%
	netcard	71	3	2.5	134	1	3.05	22%	-27%
	vga_lcd	30	2	5.56	24	2	6.41	15%	-44%
	des_perf	6	3	6.94	10	2	8.33	20%	-47%
	ethernet	12	1	12.5	10	1	12.5	0	-20%

The results show that the system clock frequency of all the cases is about several to tens of MHz for all the testbenches as shown in Figure 3.13. There are two kinds of comparisons in Table 3.2. First, ISERDES/OSERDES will be compared with Logic Multiplexing in performance. Then, the proposed algorithm will be compared with the Turki's algorithm in performance.

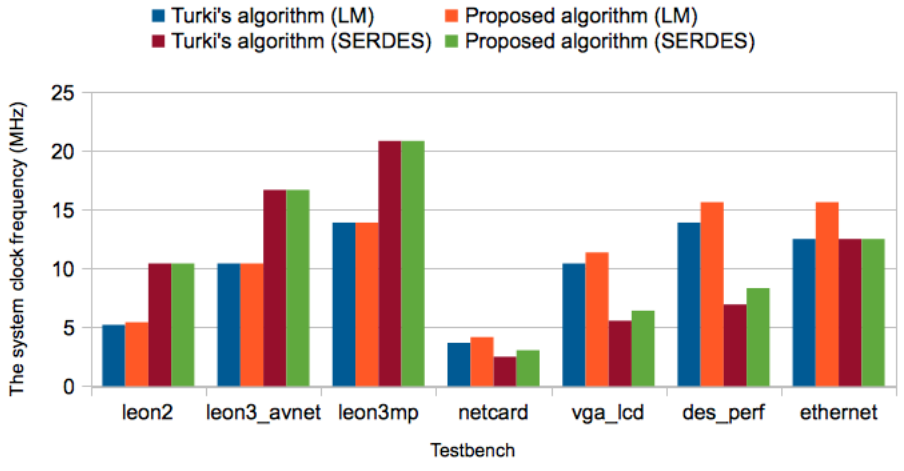


Figure 3.13: *The achieved performance in the off-the-shelf platform*

3.7.2.1 Logic Multiplexing VS ISERDES/OSERDES

According to Chapter 2, in a duo-FPGA platform, ISERDES/OSERDES can always have higher performance than Logic Multiplexing. Nevertheless, this is not the case in a multi-FPGA platform. Table 3.2 shows that, in the testbenches (netcard, vga_lcd, des_perf), the achieved performance in ISERDES/OSERDES is lower than that in Logic Multiplexing (up to -47% gain).

This is due to that, in a multi-FPGA platform, the routing hops (hop) need to be considered because the performance is influenced by both mux and hop. In Logic Multiplexing, the routing hops are realized in a pipelined way according to Section 3.6.3. Therefore, one hop has the same impact on the performance as one mux according to Equation 3.1. The existence of routing hops may increase the achieved performance due to that the routing hops can reduce mux. As a conclusion, Logic Multiplexing is a hop-friend technique. In ISERDES/OSERDES, the routing hops are not realized in a pipelined way according to Section 3.6.3. The transmission of data in the intermediate FPGA can only start when all the data is received from the Driver FPGA. Therefore, one hop has more impact on the performance than one mux according to Equation 3.2. The existence of routing hops may degrade the performance. As a conclusion, the chosen ISERDES/OSERDES implementation is a hop-hostile technique. In the testbenches (leon2, leon3_avnet, leon3mp) that have not many multi-terminal nets, the mapping process in ISERDES/OSERDES can result no routing hop and achieve higher performance than Logic Multiplexing (up to 92% gain). Nevertheless, in the testbenches (netcard, vga_lcd, des_perf) that have many multi-terminal nets, the routing hops can not be avoided in the targeted off-the-shelf platform even with ISERDES/OSERDES (hop-hostile). Therefore, in the testbenches (netcard, vga_lcd, des_perf), the achieved performance in ISERDES/OSERDES is lower than that in Logic Multiplexing (up to -47% gain).

3.7.2.2 Turki's algorithm VS proposed algorithm

Table 3.2 shows that the proposed routing algorithm can reduce the mux and hop. If Logic Multiplexing (resp. ISERDES/OSERDES) is chosen, the proposed routing algorithm can increase the performance up to 25% (resp. 22%) gain compared to the Turki's algorithm in the off-the-shelf platform by routing multi-terminal nets in multi-point tracks.

3.8 Conclusion

In this Chapter, the off-the-shelf platform is discussed and the typical implementation flow for multi-FPGA platforms is presented. The main contribution is to propose an algorithm that routes and multiplexes multi-terminal nets in multi-point tracks to spare FPGA I/Os, in order to improve the system clock frequency. Experiments are conducted using four testbenches from Gaisler Research Benchmarks and three testbenches from OpenCores Benchmarks. The results in an off-the-shelf platform of six Virtex-5 show up to 25% (resp. 22%) gain in the system clock frequency compared to the existing 2-point routing algorithm in case of Logic Multiplexing (resp.

ISERDES/OSERDES) used as inter-FPGA communication architecture.

The next Chapter focuses on the custom platform. We propose an automatic design flow for creating a custom platform. The proposed automatic design flow reduces the time-to-market of new products and lowers the entry barrier of board designers while optimizing the cost and the performance.

Chapter 4

Hardwired Custom Multi-FPGA Platform

4.1 Introduction

The previous Chapter has discussed the off-the-shelf platform and presented the typical implementation flow. The purpose of this Chapter is to discuss the custom platform and propose an automatic design flow for creating a custom platform.

The custom platform consists of a build-your-own specific multi-FPGA board, where all the inter-FPGA connections are realized using PCB traces. Different from the off-the-shelf platform that has generic and balanced connections, the connections inter FPGAs as well as the connections to external interfaces of the custom platform are user-defined and tailored for a specific design.

70% of multi-FPGA prototyping platforms are home-made custom platforms due to performance requirement (in terms of the system clock frequency), external interfaces, and cost. Indeed home-made custom platforms are tailored for a specific design and external interfaces, and the cost decreases as more platforms are needed. Nevertheless, crafting a home-made custom multi-FPGA platform is today a manual process. Only few point tools such as Cadence FSP tool [FSP, 2014] are available. Therefore, three critical aspects exist:

- Productivity: Crafting a home-made custom multi-FPGA platform is a time-consuming process (about 9 months [Sekhar, 2014]). The performance and the cost of the platform lie on the FPGA expertise and SoC DUT knowledge of the prototyping team.
- Exploration: There are many different FPGA types (i.e. vendor: Xilinx [Xilinx, 2014] or Altera [Altera, 2014], family: Virtex-7 or Stratix5, device: 2000T or GXAB, and package: FLG1925 or F1932), and different FPGA types have different logic capacity and numbers of FPGA I/Os. For a given design, the FPGA type chosen by the engineers influences the achieved performance and cost. The tradeoff between the performance and the cost exists due to that the performance of multi-FPGA platforms is limited by the inter-FPGA

communications [Amos et al., 2011]. As crafting a custom platform is a manual and time-consuming process, the board exploration with different FPGA types, which helps the engineers to design an optimum prototyping platform (cost-optimal, performance-optimal or intermediate), can not be done.

- Performance: As the ratio between the logic capacity and the number of FPGA I/Os is increasing at an exponential rate, it becomes more and more challenging to design a high-performance multi-FPGA platform.

The rest of this Chapter is organized as follows. Section 4.2 presents an example of the custom platform. Section 4.3 shows the overview of the automatic design flow for creating a custom platform. From Section 4.4 to Section 4.8, each step of the flow is detailed. Then, Section 4.9 details the proposed automatic design flow. Section 4.10 depicts the board exploration. In Section 4.11, experiments are conducted using Gaisler Research Benchmarks [Gaisler, 2014] and OpenCores Benchmarks [OpenCores, 2014]. Finally, Section 4.12 concludes this Chapter.

4.2 Platform Overview

The custom platform consists of a build-your-own specific multi-FPGA board, where all the inter-FPGA connections are realized using PCB traces. The advantage of the custom platform is that it is tailored for a specific design.

4.2.1 Specific Design

In Figure 4.1, a custom platform is created for a specific design. The design contains the following features:

- Processor
- Floating-point unit (FPU) and custom co-processor (CP)
- Separate instruction and data cache
- AHB and APB on-chip buses
- 8/16/32-bits memory controller for DDR
- On-chip peripherals such as UART, Timer, interrupt controller (IrqCtrl) and 16-bit I/O port
- Advanced on-chip debug support unit and trace buffer
- Local RAMs
- Low complexity 32-bit PCI target-only interface

- 10/100 Ethernet MAC
- Memory management unit (MMU)

The memory controller, UART, Timer, interrupt controller (IrqCtrl), 16-bit I/O port, debug serial link, Ethernet and PCI interact with peripherals. These peripherals, which are outside FPGAs but on the board, are called external interfaces.

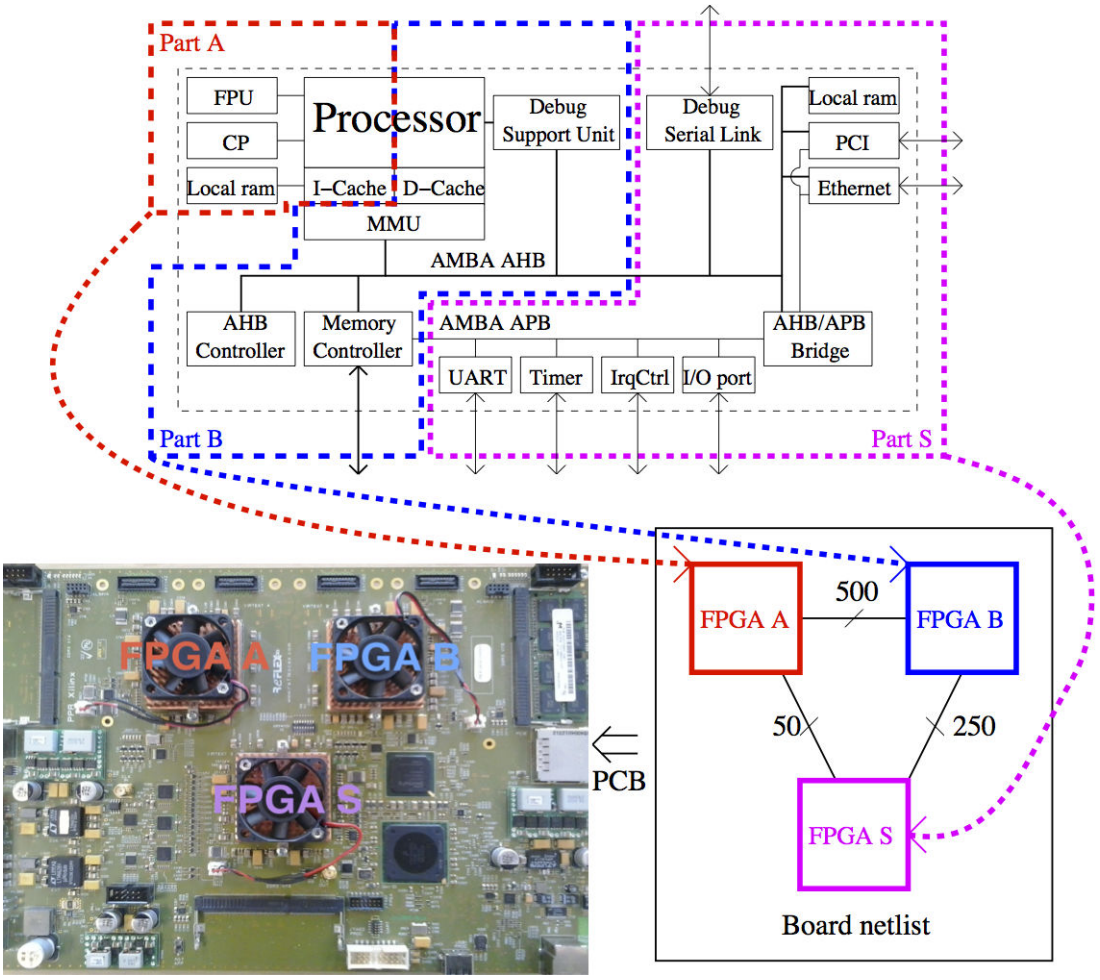


Figure 4.1: *Create a Custom Platform for a specific design*

4.2.2 Typical Flow for Creating a Custom Platform

After studying the design features, the typical flow for creating a custom platform is presented as follows.

1. Estimate the number of FPGAs required.

   The logic capacity of the design is bigger than the largest FPGA available. Therefore, three of them are needed (noted as FPGA A, FPGA B and FPGA S) and the design is partitioned

into three parts (noted as Part A, Part B and Part S). Each part's capacity fits in a single FPGA.

2. Floorplan FPGAs and external interfaces on the Printed Circuit Board (PCB).

   FPGAs and external interfaces are placed in the PCB layout with considering their possible interconnections. The positions of FPGAs and external interfaces need to ensure that the propagate delay of their possible interconnections on the board can fulfil the specifications (i.e. inter-FPGA delay should be $1ns$). Meanwhile, the DRC rules of PCB layout (such as distance and size) need to be respected.

3. Define the FPGA connectivity (tracks inter FPGAs and tracks to external interfaces).

   The advantage of the custom platform is that it is tailored for a specific design. In the example, there are many signals crossing from Part A to Part B but only few from Part A to Part S. Therefore, more inter-FPGA tracks are allocated between FPGA A and FPGA B than between FPGA A and FPGA S. In views of external interfaces, there is only 1 DDR in the example. Therefore, 1 DDR is reserved only for FPGA B that contains the memory controller. Different from the off-the-shelf platform, which has generic and balanced connections as shown in Figure 3.1, tracks of the custom platform are user-defined and tailored for the given design to achieve higher performance. Different from the off-the-shelf platform, which has 1 DDR reserved for each FPGA as shown in Figure 3.1 and wastes FPGA I/Os due to that there is only 1 DDR in the example, the tracks to external interfaces of the custom platform are tailored for the given design and no FPGA I/O is wasted. Therefore, the design partitioning is facilitated due to that the DDR is connected to the FPGA where the memory controller logic is placed in the custom platform, thus higher performance can be achieved.

4. Select FPGA I/Os. The I/O DRC rules (such as clock capability, clock region and I/O standard voltage reference levels) are considered, when choosing I/Os for each track. Then, a board netlist and a PCB layout are generated. This step can be done by the point tool Cadence FSP [Cadence, 2011].

   When choosing ISERDES/OSERDES as inter-FPGA communication architecture, there is the bank-to-bank limitation in selecting FPGA I/Os used for inter-FPGA communication according to Chapter 2.3.3. In the bank-to-bank limitation, all the FPGA I/Os need to be in LVDS, which is a signaling standard that provides high-speed data transfers by using a pair of FPGA I/O pins. And the inter-FPGA fast clock is propagated from the source FPGA to the destination FPGA (source-synchronous). According to [SelectIO, 2014], a bank is a group of FPGA I/O pins that share a common resource such as one power supply or one output current reference. Several parallel ISERDES/OSERDES can be instantiated together with only propagating one clock, when the outputs of the source FPGA (resp. the inputs of the destination FPGA) are the FPGA I/O pins from the same bank. If not, one pair of pins per bank between FPGAs is reserved to propagate the clock instead of the user data due to

FPGA technology limitation. The pair of FPGA I/Os that receives the clock need to be clock capable.

5. Map the SoC/ASIC design on this latter board, meaning that the design is partitioned and routed into each FPGA.

   The input design is synthesized targeting FPGAs (from RTL to design netlist). As the design is bigger than the FPGA, the design netlist is partitioned into several parts according to the number of FPGAs. Each part's capacity fits in a single FPGA. The signals crossing design's parts located in different FPGAs are called cut nets. Then, the cut nets are routed meaning that a cut net is allocated to an inter-FPGA track in the custom platform. As there are fewer available inter-FPGA tracks than the number of cut nets, several cut nets need to be multiplexed and sent together onto a single track on the platform.

6. Run the PnR Compiler with the design sub-netlists to generate the bitstreams for each FPGA. Then, the generated bitstreams are downloaded into the platform to model the design.

   The logic elements in each design sub-netlist are translated and placed in the corresponding logic elements in each FPGA. FPGA internal routing resources are chosen to connect the placed logic elements inside one FPGA. Then, the bitstreams are generated for each FPGA and are downloaded into the platform to model the design.

4.2.3 Problem

The custom platform is built, tailored to a specific design. As discussed in Section 2.2.2 of the state of the art, crafting a custom platform is today a manual and iterative process, thus is time-consuming (about 9 months [Sekhar, 2014]). Only few point tools such as Cadence FSP tool [FSP, 2014] are available. The performance of multi-FPGA platforms depends on the inter-FPGA data rate and tracks distribution. The data rate for the off-the-shelf platform is high (~ 1 Gbps) due to that it is ready-made by FPGA experts in commercial companies [Protium, 2014] [DINI, 2014]. For the custom platform done by the in-house team, the data rate depends on the designers' experience on how to design the high-speed signaling, high performance multiplexing and so forth. The tracks distribution for the off-the-shelf platform is generic and balanced as shown in Figure 3.1. For the custom platform done by the in-house team, it depends on the designers' experience on how to distribute tracks in order to achieve higher performance. Naturally, the custom platform should achieve higher performance than the off-the-shelf platform because it is tailored for the given design. Unfortunately, we can see that the performance of the custom platform heavily depends on designs' experience because designing such platform is still a manual process. Therefore, we often see not so good performance platforms.

4.3 The Overview of the Automatic Design Flow

In this manuscript, we propose an automatic design flow of creating a custom multi-FPGA platform for a specific design by only choosing an FPGA type. The design flow overview is shown in Figure 4.2. The flow takes the design RTL as inputs, through several steps, and generates a high performance board netlist with considering the constraints of high-speed signaling, high-performance multiplexing and so forth. After that, the flow implements the design into the generated board and generates bitstreams for each FPGA. The outputs of the flow is the board netlist and the design sub-netlists. The flow can be divided into 6 main steps and detailed as follows.

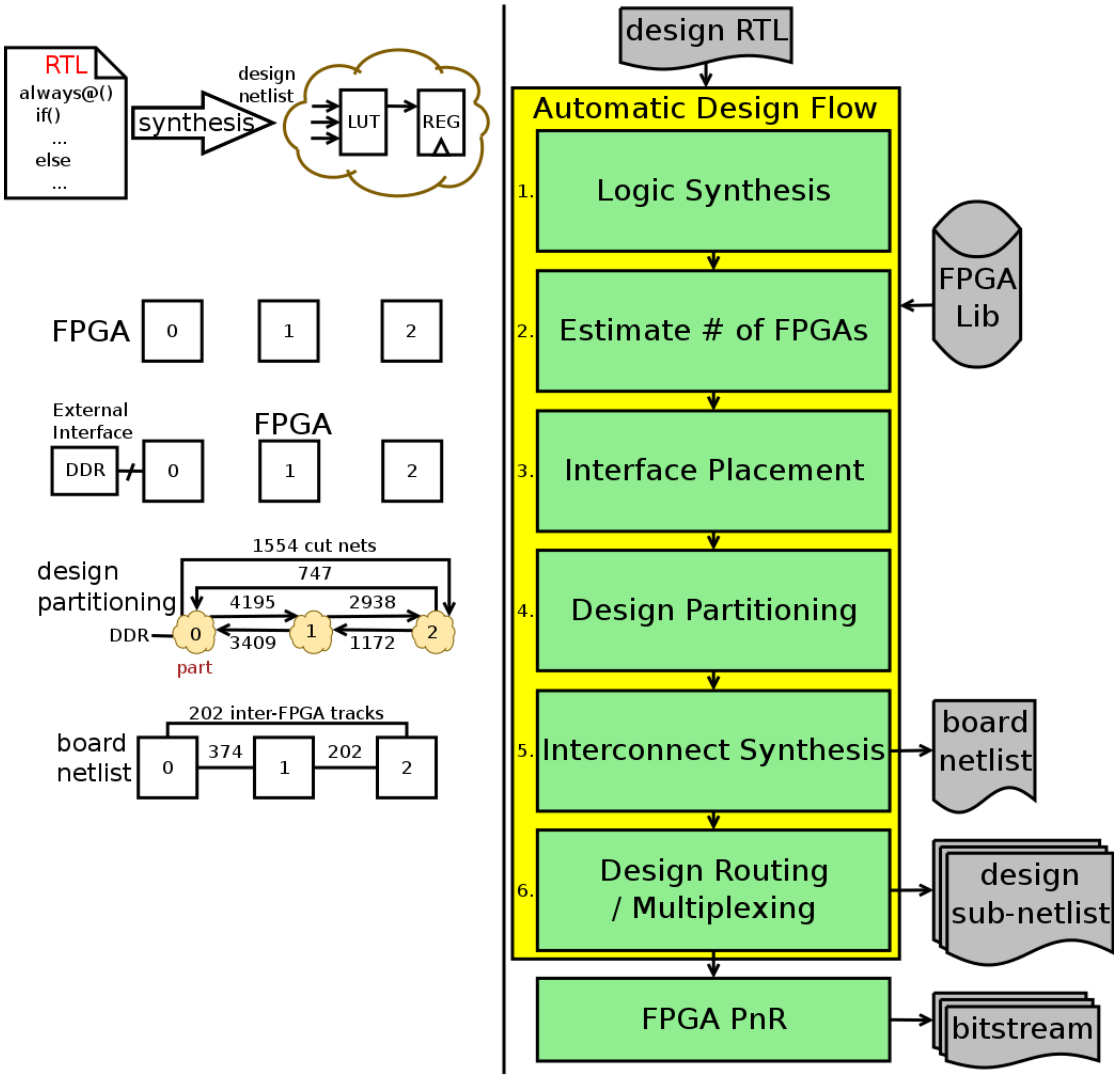


Figure 4.2: *The design flow overview for creating a custom platform*

1. Logic Synthesis: The input design is synthesized targeting FPGAs (from RTL to design netlist).

2. Estimate the number of FPGAs: The minium number of FPGAs required is estimated in

the platform (according to the logic capacity of the input design, the logic capacity of the chosen FPGA, and the maximum FPGA logic capacity utilization).

3. Interface Placement: External interfaces to FPGAs are connected.
4. Design Partitioning: As the SoC/ASIC design is bigger than the FPGA, the design netlist is partitioned into several parts according to the number of FPGAs. Each part's capacity fits in a single FPGA. The signals crossing design's parts located in different FPGAs are called cut nets.
5. Interconnect Synthesis: Inter-FPGA tracks are distributed according to the distribution of cut nets and a board netlist is generated.
6. Design Routing / Multiplexing: The cut nets are routed meaning that a cut net is allocated to an inter-FPGA track in the custom platform. As there are fewer available inter-FPGA tracks than the number of cut nets, several cut nets need to be multiplexed and sent together onto a single track on the platform.

Before launching the automatic design flow, the FPGA Lib need to be established. The chosen FPGA for the automatic design flow need to be contained in the FPGA Lib.

4.3.1 FPGA Lib

In the FPGA Lib, there are hundreds of different FPGAs, which have different types depending on the vendor (i.e. Xilinx [Xilinx, 2014] or Altera [Altera, 2014]), the family (i.e. Virtex-7 or Stratix5), the device (i.e. 2000T or GXAB), and the package (i.e. FLG1925 or F1932). Different types of FPGAs have different logic capacity, different I/O capacity and different prices. An example of five different Xilinx FPGA types are shown in Table 4.1.

Table 4.1: Logic capacity, I/O capacity and estimated price of different FPGA types

FPGA type			Logic Capacity				I/O Capacity			Price
Family	Device	Package	LUT	REG	RAM	DSP	I/Os	LVDS	bank	($)
Virtex-5	LX330	FF1760	207,360	207,360	288	192	1,200	600	32	2,310
Spartan-6	LX150T	FG900	92,152	184,304	134	180	540	270	6	1,050
Virtex-6	LX550T	FF1759	343,680	687,360	632	864	840	420	21	3,850
Virtex-6	LX760	FF1760	474,240	948,480	720	864	1,200	600	30	5,320
Virtex-7	2000T	FLG1925	1,221,600	2,443,200	1,292	2,160	1,200	576	24	14,000

- Logic Capacity (represented by the number of logic elements contained in the FPGA). There are four kinds of logic elements which are taken into consideration: Look-Up Table (noted as LUT), Register (REG), Block RAM (RAM), and Embedded DSP (DSP).
- I/O Capacity (represented by the number of I/Os, the number of I/O pairs in LVDS and the number of banks contained in the FPGA).
- Estimated Price (7 $ per ~ 1000 logic elements).

4.4 Logic Synthesis

The first step of the proposed automatic design flow is to synthesize the input design targeting FPGAs, which translates the design from RTL to netlist. After the logic synthesis, the logic capacity of the input design is represented by the number of logic elements (LUT, REG, RAM, and DSP) contained. This step can be realized by Xilinx XST Synthesis tool [XST, 2014], Altera Quartus Synthesis tool [Quartus, 2014], or third party tools such as Synopsys Synplify tool [Synplify, 2014] and Mentor Graphics Precision tool [Precision, 2014].

4.5 Estimate the number of FPGAs

After the logic synthesis, the next step is to estimate the minimum number of FPGAs required in the platform according to the logic capacity of the input design, the logic capacity of the chosen FPGA type and the maximum FPGA logic capacity utilization. The logic elements in the prototyped design (resp. in the chosen FPGA) are noted as: LUT_{design}, REG_{design}, RAM_{design} and DSP_{design} (resp. LUT_{FPGA}, REG_{FPGA}, RAM_{FPGA} and DSP_{FPGA}). The logic capacity of one FPGA can not be fully used due to the intra-routability, possible product upgrades of the prototyped design and etc. The maximum FPGA logic capacity utilization is noted as F. The minimum number of FPGAs required in the platform (noted as NUM_{FPGA}) is shown in Equation 4.1, which is the ceiling value of the largest ratio of logic elements in the prototyped design to the available logic elements in the target FPGA. If the prototyped design is DSP-intensive, it may be more beneficial to choose an FPGA with more DSPs.

$$NUM_{FPGA} = \lceil max(\frac{LUT_{design}}{F * LUT_{FPGA}}, \frac{REG_{design}}{F * REG_{FPGA}}, \frac{RAM_{design}}{F * RAM_{FPGA}}, \frac{DSP_{design}}{F * DSP_{FPGA}}) \rceil \tag{4.1}$$

When the logic elements in the prototyped design netlist exceeds the logic capacity of the chosen FPGA multiplied by the maximum FPGA logic capacity utilization, the multi-FPGA platform is needed. In this case, the minimum number of FPGAs required in the platform need to be estimated. This step can be realized by the commercial tool waCore provided by the company Flexras Technologies [Flexras, 2014]. As a result, an XML file that contains the information of the number of FPGAs required in the platform is generated.

4.6 External Interface Placement

One of the advantages for FPGA-based prototyping is to enable "real world" testing, while the prototyped designs are put into actual hardware and real external interfaces (such as DDR, PCI,

Ethernet and etc.) are used. After estimating the number of FPGAs in the platform, the next step is to connect external interfaces to FPGAs.

There are many possibilities to connect external interfaces to FPGAs. Different possibilities result different results in performance. In order to achieve higher performance, an algorithm (either exhaustive or heuristic) need to be proposed to optimize the placement of external interfaces. The exhaustive method can find the best performance by testing all the possibilities. While the number of tests is enormous (i.e. NP-hard problems), heuristic methods need to be implemented to achieve the optimal performance. As the number of external interfaces in a prototyped design is usually about several to dozens, the number of possibilities is limited. Therefore, we propose the External Interface Placement Algorithm in this manuscript (as depicted in Algorithm 1) to exhaust all the possibilities. The proposed external interface placement algorithm is realized in python programming language.

Algorithm 1 External Interface Placement Algorithm

Require:
The number of FPGAs, num_fpga;
The number of External Interfaces (XIs), num_inter;
XI[$num\_inter - 1$...0] (XI[0] = 0 means that the XI 0 is connected to FPGA 0);
Ensure:
if $num\_fpga > num\_inter$ {the maximum number of FPGAs connected to XIs: num} **then**
$num = num\_inter$
else
$num = num\_fpga$
end if
for $i \in (0, num]$ {the number of FPGAs connected to XIs: i} **do**
if $i == 1$ {i=1, only one FPGA is connected to XIs} **then**
Place_all_the_XIs_to_the_same_FPGA($XI[...]$)
else if $i == num\_inter$ **then**
Place_each_XI_to_a_different_FPGA($XI[...]$, num_inter)
else
Other_case(XI[...], i)
end if
end for
return Choose the case with the highest achieved performance

In the Algorithm, different placements of external interfaces can be classified into three different types of cases: all the external interfaces connected to the same FPGA, each external interface connected to a different FPGA, and other cases where at least two FPGAs are used and at least one FPGA is connected with several external interfaces. If the placement is feasible, each case will be implemented respectively by the rest steps of the automatic design flow (presented in Section 4.3) to evaluate the achieved performance. A feasible placement means that the available FPGA I/Os after assigning the XIs are still more than 0 for each FPGA. Finally, the automatic design flow chooses the case that achieves the highest performance. The number of FPGAs in the multi-FPGA platform is noted as num_fpga. The number of external interfaces is noted as num_inter. The

table XI[...] represents the placement of external interfaces (i.e. XI[0] = 0 means that the external interface numerated as 0 is connected to the FPGA 0). The maximum number of FPGAs connected to external interfaces is noted as num. If $num\_fpga > num\_inter$, num will equal num_inter. If not, num will equal num_fpga. The number of FPGAs connected to external interfaces is noted as i.

- If $i = 1$, it means that all the external interfaces will be placed in one FPGA, therefore the algorithm will go to the function Place_all_the_XIs_to_the_same_FPGA().
- If $i = num\_inter$, it means that each external interface will be placed in a different FPGA, therefore the algorithm will go to the function Place_each_XI_to_a_different_FPGA().
- If $1 < i < num\_inter$, it means that at least two FPGAs are used and at least one FPGA is connected with several external interfaces, therefore the algorithm will go to the function Other_case().

An example, where num_fpga is 4 and num_inter is 3, will be studied in this Section. In the example, 4 FPGAs are numerated as FPGA 0, FPGA 1, FPGA 2 and FPGA 3. 3 external interfaces are numbered as (XI[2], DDR), (XI[1], Ethernet) and (XI[0], PCI). The Ethernet is shortened as Eth in the following figures.

4.6.1 All the External Interfaces connected to the same FPGA

When all the External Interfaces are connected to the same FPGA, the number of different cases depends on the number of FPGAs in the platform (num_fpga), thus is num_fpga cases. Figure 4.3 shows that the results of these cases are the same due to the symmetry. Therefore, only one case is executed to reduce the runtime. In the example, there are 4 different cases because there are 4 FPGAs in the platform. Nevertheless, there is only one case that will be executed.

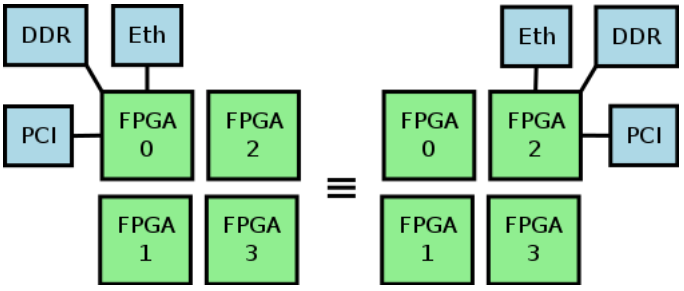


Figure 4.3: *All the External Interfaces connected to the same FPGA*

The function Place_all_the_XIs_to_the_same_FPGA() is depicted in Algorithm 2 and all the external interfaces are connected to FPGA 0.

Algorithm 2 Function: Place_all_the_XIs_to_the_same_FPGA()

Require:
XI[$num\_inter - 1$...0];
Ensure:
for $k \in [0, num\_inter)$ **do**
XI[k] = 0
end for
if $feasible\_placement() == 1$ **then**
The rest steps of the automatic design flow()
end if
return The achieved performance

4.6.2 Each External Interface connected to a different FPGA

When each External Interface is connected to a different FPGA, the number of different cases depends on the number of FPGAs in the platform (num_fpga) and the number of external interfaces in the prototyped design (num_inter), thus is $A^{num\_inter}_{num\_fpga}$ cases. Figure 4.4 shows that the results of these cases are the same due to the symmetry. Therefore, only one case is executed to reduce the runtime. In the example, there are 24 different cases because there are 4 FPGAs in the platform and 3 XIs in the prototyped design. Nevertheless, there is only one case that will be executed.

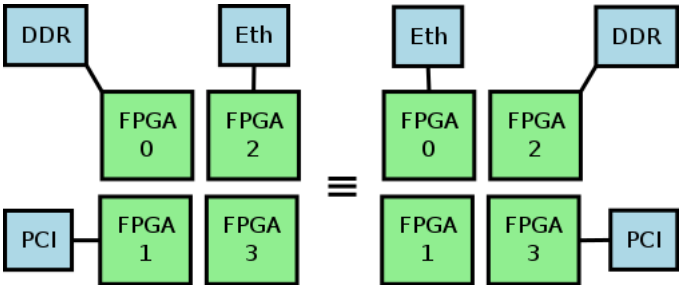


Figure 4.4: *Each External Interface connected to a different FPGA*

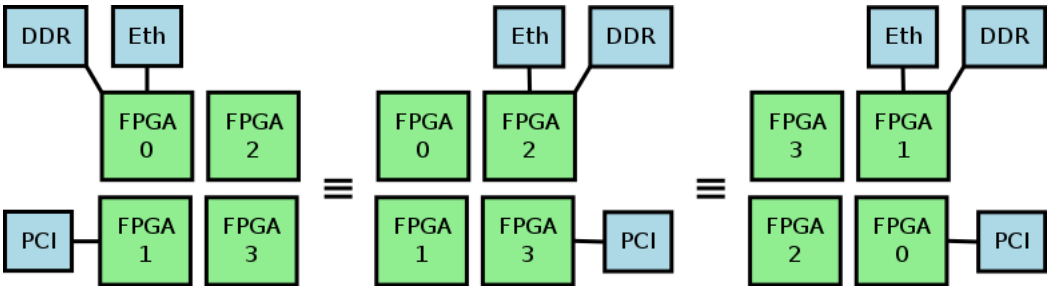
The function Place_each_XI_to_a_different_FPGA() is depicted in Algorithm 3 and the external interface XI[k] is connected to FPGA k.

Algorithm 3 Function: Place_each_XI_to_a_different_FPGA()

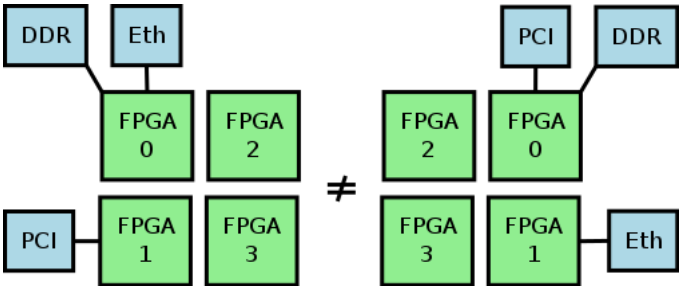
Require:
XI[...];
Ensure:
for $k \in [0, num\_inter)$ **do**
XI[k] = k
end for
if $feasible\_placement() == 1$ **then**
The rest steps of the automatic design flow()
end if
return The achieved performance

4.6.3 Other cases

In the other cases, where at least two FPGAs are used and at least one FPGA is connected with several External Interfaces, the number of different cases depends on the number of FPGAs connected to external interfaces (num) and the number of external interfaces in the prototyped design (num_inter), thus is $pow(num, num\_inter)$ cases. There can be some identical cases (i.e. PCI in one FPGA, DDR and Ethernet in another FPGA as shown in Figure 4.5(a)), but not all the cases are the same (i.e. there may be other case such as Ethernet in one FPGA, PCI and DDR in another FPGA as shown in Figure 4.5(b)). In the other cases, all the possible placements are swept and the identical cases due to the symmetry are executed only one time. In the example, there will be $pow(2, 3) = 8$ cases in total due to that there are 3 external interfaces and 2 FPGAs connected to external interfaces.



(a) symmetry: identical case



(b) asymmetry: different case

Figure 4.5: *Other cases*

The function Other_case() is depicted in Algorithm 4. If the number of FPGAs connected to external interfaces is noted as num, the external interfaces will only be connected to the FPGAs indexed between (0, $num - 1$) to remove the symmetry. In the example, the number of FPGAs connected to external interfaces is 2. Therefore, the external interfaces will only be connected to the FPGAs indexed between (0, 1) to remove the symmetry. In the example, the case as shown at the middle of Figure 4.5(a) that DDR and Ethernet are placed in FPGA 2 and PCI is placed in FPGA 3 is the same with the case as shown at the left of Figure 4.5(a) that DDR and Ethernet are placed in FPGA 0 and PCI is placed in FPGA 1.

Algorithm 4 Function: Other_case()

Require:
i: the number of FPGAs connected to XIs;
XI[...];

Ensure:
for $j \in [0, pow(i, num\_inter))$ {If the number of FPGAs connected to XIs is i, there will be pow(i, num_inter) cases in total} **do**
for $k \in [0, num\_inter)$ **do**
XI[k] = ((j / pow(i, k)) % i) {Get one placement for one XI[k] basing from the parameter i, j, k, permitting to sweep all the possible placement for all the XIs. This formula (especially by %i) will permit to index the FPGA from the minimum index to remove the symmetry.}
end for
temp[...] {Define a temp value, to identify that the assignment is correct or not}
asymmetry = 1 {Flag of the asymmetry solution}
for $k \in [0, num\_inter)$ {Sweep the placement results of all the XIs} **do**
same = 0 {Flag of two XIs connected to the same FPGA}
if k == 0 **then**
Insert XI[0] into temp[...]
else
for $x \in [0, number\ of\ elements\ in\ temp[...])$ **do**
if temp[x] == XI[k] {the new XI is connected to the same FPGA that the previous swept XI has been connected with} **then**
same = 1
end if
end for
if same == 0 {the new XI is connected to a new FPGA} **then**
for $x \in [0, number\ of\ elements\ in\ temp[...])$ **do**
if XI[k] < temp[x] {If the new placed XI is connected to an FPGA with less index number, this case will be removed due to the symmetry.} **then**
asymmetry = 0
end if
end for
Insert XI[k] into temp[...]
end if
end if
end for
if the number of elements in temp[...] != i {the number of elements in temp is not equal the number of FPGAs connected to XIs} **then**
asymmetry = 0
end if
if $asymmetry == 1\ and\ feasible\_placement == 1$ **then**
The rest steps of the automatic design flow()
end if
end for
return Choose the case with the highest achieved performance

The external interfaces are placed one by one from XI[0] to XI[$num\_inter - 1$]. The newly

placed External Interface XI[k] will be connected to an FPGA with no less index number than the previously placed external interfaces XI[k-1] (i.e. XI[k] is placed to FPGA x, XI[k-1] is placed to FPGA y and $x \geq y$). In the example, (XI[0], PCI), (XI[1], Ethernet), (XI[2], DDR). If PCI is placed in one FPGA, DDR and Ethernet are placed in another FPGA, PCI will be placed in FPGA 0 and DDR and Ethernet will be placed in FPGA 1 as shown at the right of Figure 4.5(a). In the example, the case as shown at the left of Figure 4.5(a) that PCI is placed in FPGA 1 and DDR and Ethernet are placed in FPGA 0 is the same with the case as shown at the right of Figure 4.5(a). If the cases are not identical, they need to be executed to obtain the result. Finally, the selected placement is the one with the highest performance.

4.6.4 Conclusion

In this Section, we proposed an external interface placement algorithm (that is realized in python programming language). Different placements of external interfaces can be classified into three different types of cases: all the external interfaces connected to the same FPGA, each external interface connected to a different FPGA, and other cases where at least two FPGAs are used and at least one FPGA is connected with several external interfaces. Finally, the automatic design flow chooses the case that achieves the highest performance. As a result, the FPGA-Interface tracks file in XML format is generated.

4.7 Design Partitioning

After the external interface placement, the next step is the design partitioning. As discussed in Section 3.5, the design partitioning in the multi-FPGA prototyping is a NP-hard problem [Garey and Johnson, 1990]. The partitioning needs to take the connection constraints (the distribution of inter-FPGA connections) into consideration, in order to achieve a higher performance solution compared to the partitioning without any board information. In this manuscript, the design partitioning is done by the commercial tool waPart provided by the company Flexras Technologies [Flexras, 2014]. Nevertheless, the custom platform with the distribution information of inter-FPGA connections is not available at this moment. Therefore, before partitioning the design, a temporary balanced platform need to be generated. Then, the partitioning tool takes the connection constraints and the size constraints (the number of logic elements in one FPGA) of the temporary balanced platform into consideration. The objective of the design partitioning is to minimize the total number of cut nets, which contributes to increase the performance of multi-FPGA prototyping.

The generation of the board netlist for the temporary balanced platform is detailed in Figure 4.6 (realized in python programming language). It takes the XML file of FPGA-Interface tracks as input. This XML file is generated by the external interface placement algorithm and contains the information that which external interface will be connected to which FPGA. According to the chosen inter-FPGA communication architecture, the available I/Os per FPGA are different. In

Chapter 2, we have detailed two existing inter-FPGA communication architectures. The first one is Logic Multiplexing that uses logic blocks of the FPGA fabric to implement the multiplexing for inter-FPGA communication. Another is ISERDES/OSERDES that uses dedicated output parallel-to-serial converters (OSERDES) and input serial-to-parallel converters (ISERDES) with LVDS for inter-FPGA communication. If Logic Multiplexing (resp. ISERDES/OSERDES) is used as inter-FPGA communication architecture, the available I/Os are the number of FPGA I/Os (resp. the number of FPGA I/O pairs in mode LVDS). The first step of the algorithm is to reserve FPGA I/Os for global signals (i.e. global clocks, global reset), reset chain and external interfaces. Then, the temporary balanced platform netlist in Verilog format will be generated.

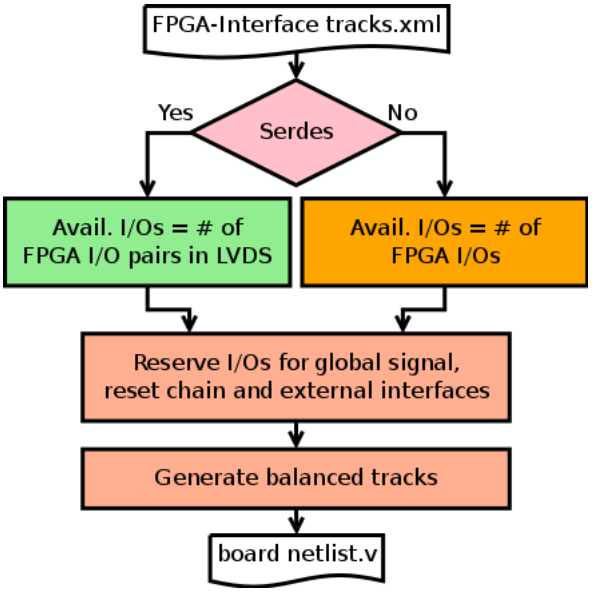


Figure 4.6: *The temporary balanced platform generation algorithm*

An example, which uses three FPGAs to create the temporary balanced platform for an industrial design, is studied in this Section. The generation of the temporary balanced platform is independent of FPGA types. The chosen FPGA in this example is the XC7V2000TFLG1925 (family: Virtex-7, device: 2000T, and package: FLG1925, thus FPGA type: XC7V2000TFLG1925) that is the largest FPGA from Xilinx [Xilinx, 2014]. If Logic Multiplexing (resp. ISERDES/OSERDES) is chosen as the inter-FPGA communication architecture, the maximum I/Os per FPGA are 1200 FPGA I/Os (resp. 576 pairs of FPGA I/Os in mode LVDS due to that several FPGA I/Os are not LVDS capable).

4.7.1 Reservation of I/Os for Global Signals, Reset Chain and External Interfaces

4.7.1.1 Global Signals

The Global Signals are peripherals that connect to all the FPGAs in the platform. The most common Global Signals are the PLL (affords global clocks), and the Reset Button (affords the global

reset) as shown in Figure 4.7.

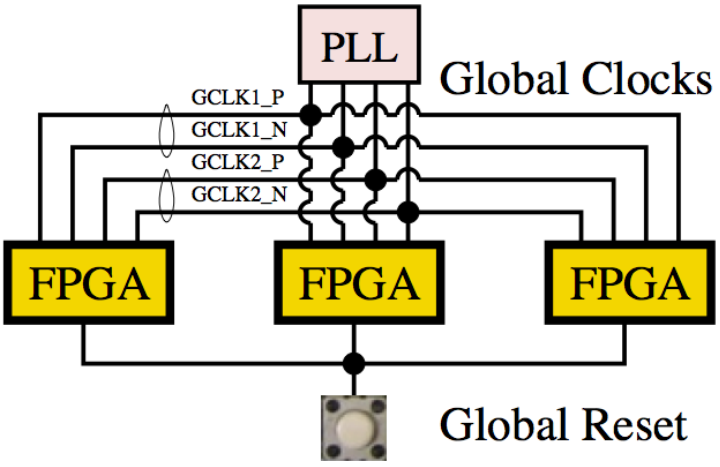
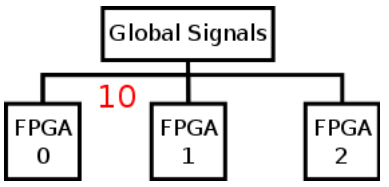
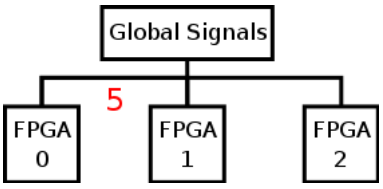


Figure 4.7: *Global signals (i.e. global clock and global reset)*

In the example, 10 FPGA I/Os in case of Logic Multiplexing (resp. 5 pairs of FPGA I/Os in mode LVDS in case of ISERDES/OSERDES) will be reserved for global signals in all the FPGAs as depicted in Figure 4.8(a) (resp. as depicted in Figure 4.8(b)). After that, available I/Os per FPGA are 1190 (resp. 571) as depicted in Figure 4.8(c).



(a) the platform (I/Os)



(b) the platform (LVDS I/Os)

FPGA	I/Os	LVDS I/Os
FPGA 0	1190	571
FPGA 1	1190	571
FPGA 2	1190	571

(c) available I/Os

Figure 4.8: *The platform after assigning global signals*

Comparison with multi-point tracks The difference between Global Signal tracks and multi-point tracks is depicted in Figure 4.9. The nets passing through global signal tracks have the driver (or the receiver) to a peripheral outside FPGAs, while the driver and all the receivers of the nets passing through multi-point tracks are in FPGAs.

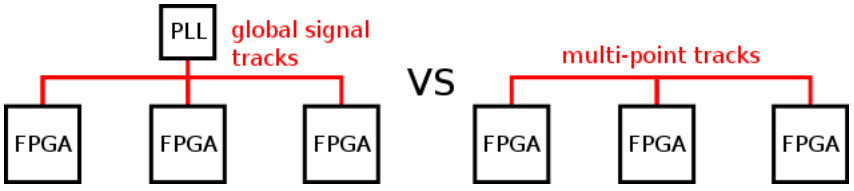
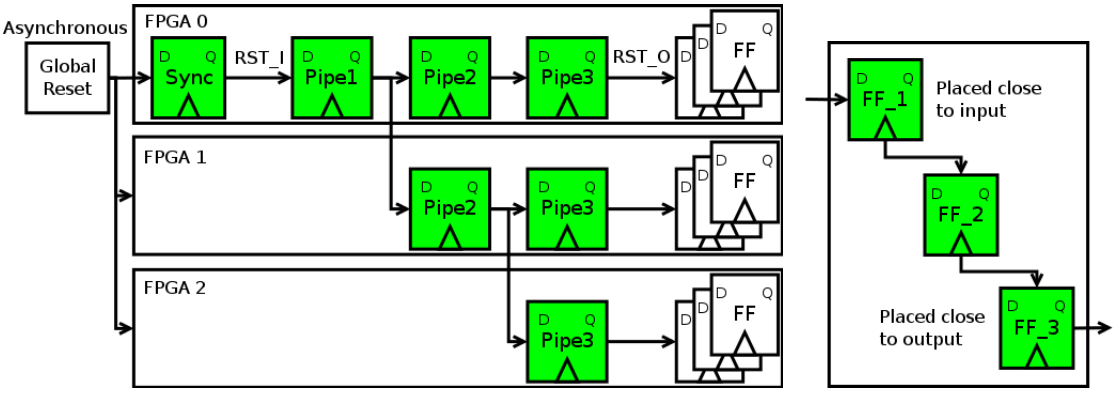


Figure 4.9: *Global signal tracks VS multi-point tracks*

4.7.1.2 Reset Chain

According to [Amos et al., 2011], the FPGAs in the platform need to be reseted at the same time, which is called reset synchronization. The asynchronous global reset enters in each FPGA and is captured by a flip-flop (FF) in order to be synchronous to the system clock edge. As the delay of the Interface global reset to different FPGAs can be different (i.e. the delay of the Interface global reset to FPGA 2 is larger than the delay of the Interface to FPGA 0), there is a risk that the reset is captured in different system clock edges in different FPGAs. The reset chain architecture as depicted in Figure 4.10(a) is proposed in [Amos et al., 2011] to resolve this problem. In the architecture, the global reset enters in one FPGA and then propagated to other FPGAs by inter-FPGA tracks. The Pipe1 is not replicated because the synchronization of the incoming global reset has to be done in only one place. The output of Pipe1 in FPGA 0 will drive two Pipe2: one in FPGA 0 and another in FPGA 1. The reset will be captured by these two Pipe2 in the same system clock edge due to that the inter-FPGA communication architectures (Logic Multiplexing and ISERDES/OSERDES) presented in Chapter 2.3 assure that the transmission of the data will be finished in one system clock cycle. Nevertheless, there is still a possibility that the pipeline stage in each FPGA would introduce delay and reduce the performance because if there is one Flip-Flop (noted as FF) in each stage, then it might be placed near the input pad, the output pad or anywhere in between; this might also be different in each FPGA. Therefore, the reset chain pipeline stage using three FFs is shown in Figure 4.10(b).

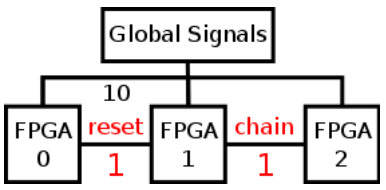


(a) The architecture of the reset chain (b) The pipeline using three FFs

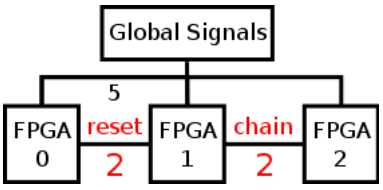
Figure 4.10: *The reset chain*

In the reset chain architecture, 1 FPGA I/O in case of Logic Multiplexing is reserved for the reset chain in FPGA i and FPGA i+1 ($0 <= i <= number\ of\ FPGAs - 2$) as shown in Figure 4.11(a). Nevertheless, in case of ISERDES/OSERDES, another pair of FPGA I/Os need to be reserved for sending the clock when sending the reset chain due to that ISERDES/OSERDES is source synchronous inter-FPGA communication architecture presented in Section 2.3.3. Therefore, 2 pairs of FPGA I/Os in mode LVDS in case of ISERDES/OSERDES are reserved for the reset chain in FPGA i and FPGA i+1 ($0 <= i <= number\ of\ FPGAs - 2$) as shown in Figure 4.11(b). After assigning the FPGA I/Os for the reset chain, available I/Os per FPGA are

shown in Figure 4.11(c).



(a) the platform (I/Os)



(b) the platform (LVDS I/Os)

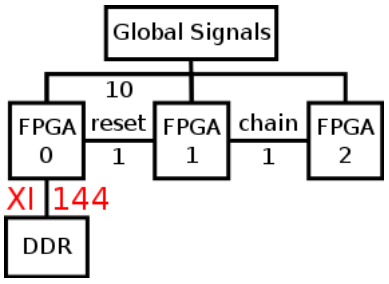
FPGA	I/Os	LVDS I/Os
FPGA 0	1189	569
FPGA 1	1188	567
FPGA 2	1189	569

(c) available I/Os

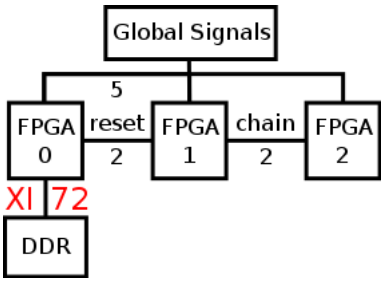
Figure 4.11: *The platform after assigning the reset chain*

4.7.1.3 External Interfaces

In the example, the prototyped design has 1 DDR. According to External Interface Placement Algorithm presented in Section 4.6, the DDR will be connected to the FPGA 0. Therefore, 144 FPGA I/Os in case of Logic Multiplexing (resp. 72 pairs of FPGA I/Os in mode LVDS in case of ISERDES/OSERDES) in FPGA 0, are reserved for DDR as shown in Figure 4.12(a) (resp. as shown in Figure 4.12(b)). After that, available I/Os per FPGA are shown in Figure 4.12(c).



(a) the platform (I/Os)



(b) the platform (LVDS I/Os)

FPGA	I/Os	LVDS I/Os
FPGA 0	1045	497
FPGA 1	1188	567
FPGA 2	1189	569

(c) available I/Os

Figure 4.12: *The platform after assigning the external interfaces*

In this sub section 4.7.1, the reservation of I/Os for global signals, the reset chain and external interfaces is discussed. Due to that the critical path of multi-FPGA platforms is the inter-FPGA tracks where the cut nets pass through, the tracks for global signals, the reset chain and external interfaces are not shown in the figures of the following platforms.

4.7.2 Generation of the Temporary Balanced Platform

After reserving I/Os for global signals, the reset chain and external interfaces, a temporary balanced platform will be generated. The first step is to choose the FPGA that has the minimum number of available I/Os. If the chosen FPGA is X, the number of inter-FPGA tracks between FPGA X and the other FPGA is the same for all the FPGA pairs. According to Equation 4.2, this number equals the number of available FPGA I/Os divided by the number of remaining FPGAs in the platform. In the example, the chosen FPGA is FPGA 0 and the minimum number of available

I/Os is 1045 in case of Logic Multiplexing (resp. 497 in case of ISERDES/OSERDES). Therefore, 522 in case of Logic Multiplexing (resp. 248 in case of ISERDES/OSERDES) will be distributed between FPGA 0 and FPGA 1 and between FPGA 0 and FPGA 2 as shown in Figure 4.13(a) (resp. as shown in Figure 4.13(b)).

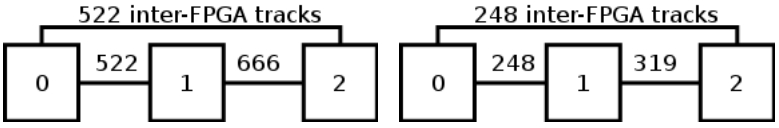
$$Tracks_{(X,\,Y)} = \frac{Avail\ I/Os\ on\ X}{number\ of\ FPGAs\ in\ the\ platform\ -\ 1} \tag{4.2}$$

(a) the balanced platform (I/Os) (b) the balanced platform (LVDS I/Os)

Figure 4.13: *The generation of the balanced platform in the custom platform design flow: Step 1*

Once inter-FPGA tracks for an FPGA are defined, the number of available I/Os for corresponding FPGAs are updated. Then, the previous step is iterated to choose one FPGA and define the corresponding inter-FPGA tracks. In the example, after having defined the inter-FPGA tracks of FPGA 0, available FPGA I/Os are updated as shown in Figure 4.14(a). In the next iteration, FPGA 1 has less available I/Os and inter-FPGA tracks of FPGA 1 will be defined. Therefore, 666 tracks in case of Logic Multiplexing (resp. 319 tracks in LVDS in case of ISERDES/OSERDES) will be defined between FPGA 1 and FPGA 2 as shown in Figure 4.14(b) (resp. as shown in Figure 4.14(c)). When all the inter-FPGA tracks are defined, the iteration will be finished and the temporary balanced platform will be generated.

FPGA	I/Os	LVDS I/Os
FPGA 0	1	1
FPGA 1	666	319
FPGA 2	667	321



(a) available I/Os (b) the balanced platform (I/Os) (c) the balanced platform (LVDS I/Os)

Figure 4.14: *The generation of the balanced platform in the custom platform design flow: Step 2*

4.7.3 Conclusion

In this Section, the design partitioning that partitions the design netlist into several parts is discussed. The first step is the generation of the temporary balanced platform by the self-developed tool (that is realized in python programming language). Then, the design netlist will be partitioned with the temporary balanced platform. The partitioning process is done by the commercial tool provided by the company Flexras Technologies [Flexras, 2014]. As a result, the partitioned design netlist file in Verilog format and the cut nets distribution file in XML format are generated.

4.8 Interconnect Synthesis

After the design partitioning, the next step is to distribute inter-FPGA tracks according to the distribution of cut nets in order to tailor the platform for the given design. We call this step the Interconnect Synthesis. In this manuscript, we propose an Interconnect Synthesis Algorithm, which is depicted in Figure 4.15. The proposed interconnect synthesis algorithm is realized in python programming language.

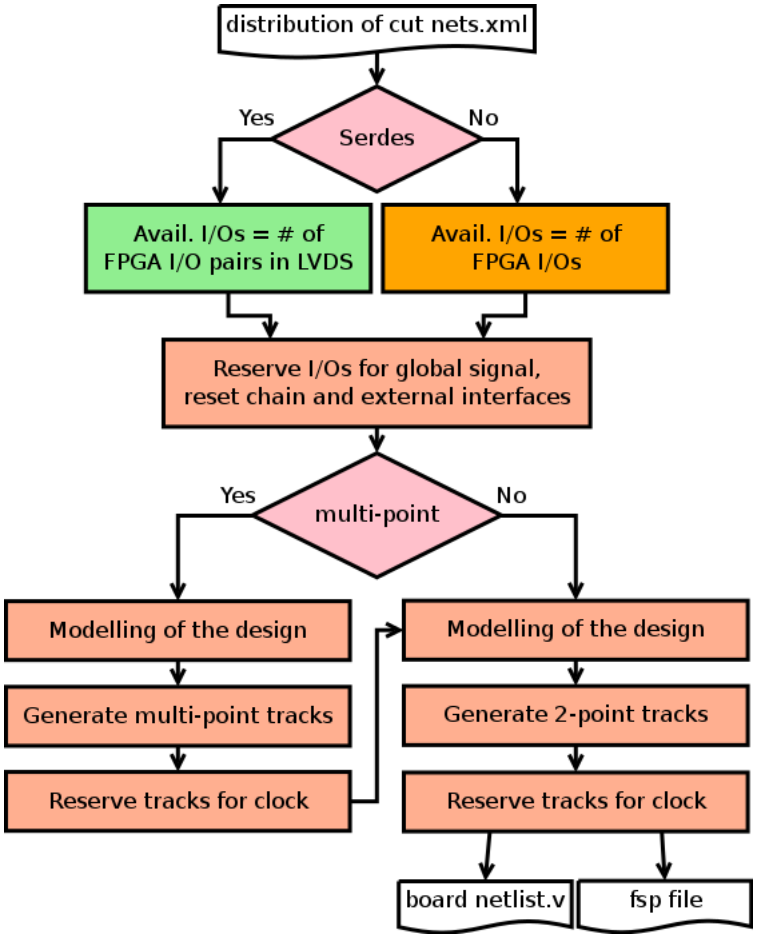


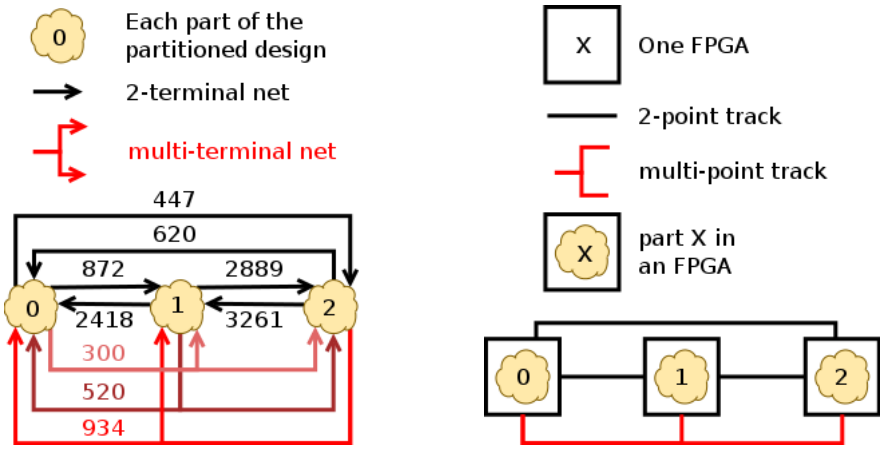
Figure 4.15: *The interconnect synthesis algorithm*

The proposed interconnect synthesis algorithm takes the cut nets distribution XML file as input. If Logic Multiplexing (resp. ISERDES/OSERDES) is used as inter-FPGA communication architecture, the available I/Os are the number of FPGA I/Os (resp. the number of FPGA I/O pairs in mode LVDS). The first step of the algorithm is to reserve FPGA I/Os for global signals (i.e. global clocks, global reset), reset chain and external interfaces. This step has been detailed in the previous Section 4.7.1. Due to that the critical path of multi-FPGA platforms is the inter-FPGA tracks where the cut nets pass through, the tracks for global signals, the reset chain and external interfaces are not shown in the figures of the following platforms. Then, the inter-FPGA tracks will be generated. There are different types of inter-FPGA tracks. The first one is 2-point tracks

that connect only two FPGAs. Another is multi-point tracks that connect more than two FPGAs. In the manuscript, multi-point tracks are assumed to connect all the FPGAs in the multi-FPGA platform. If the number of FPGAs in the platform is noted as m, multi-point tracks are m-point. The number of terminals for a cut net in the partitioned design is noted as n, varying from 2 to m. Chapter 3 shows that multi-point tracks can spare FPGA I/Os when $m \leq 2(n-1)$, and waste FPGA I/Os when $m > 2(n-1)$. Nevertheless, experiments about multi-point tracks in the off-the-shelf platform show that the number of cut nets on a multi-point track should be a half of that on a 2-point track, in order to avoid that the critical path is the multi-point track and increase the performance. In order to study whether the multi-point tracks are beneficial for custom platforms, the generated custom platforms can be classified into two categories: the platform with only 2-point tracks, and the platform with both 2- and multi-point tracks.

An example, which uses three FPGAs to create custom platforms for an industrial design, is studied in this Section. The proposed interconnect synthesis algorithm is independent of FPGA types. The chosen FPGA in this example is the XC7V2000TFLG1925 that is the largest FPGA from Xilinx [Xilinx, 2014]. If Logic Multiplexing (resp. ISERDES/OSERDES) is chosen as the inter-FPGA communication architecture, the maximum I/Os per FPGA are 1200 FPGA I/Os (resp. 576 pairs of FPGA I/Os in mode LVDS due to that several FPGA I/Os are not LVDS capable).

After reserving I/Os for global signals, the reset chain and external interfaces, the next step in the generation of the multi-FPGA platform is to distribute inter-FPGA tracks. The inter-FPGA tracks are distributed according to the distribution of cut nets in the partitioned design. In the example, the partitioned design is shown in Figure 4.16(a). There are both 2- and multi-terminal nets in the partitioned design. The multi-FPGA platform to be generated is shown in Figure 4.16(b). There are two types of inter-FPGA tracks: 2-point and multi-point tracks. Therefore, there are two categories of custom multi-FPGA platforms according to the types of inter-FPGA tracks: the platforms with only 2-point tracks and the platforms with both 2- and multi-point tracks.



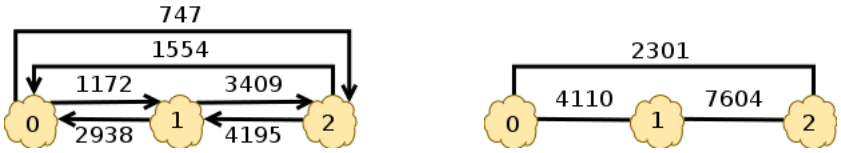
(a) partitioned design (b) platform to be generated

Figure 4.16: *An example of the partitioned design*

4.8.1 Generation of the Custom Platform with Only 2-Point Tracks

4.8.1.1 Design Modelling

Multi-terminal nets will be split in several 2-terminal nets which have the same driver but different receivers as shown from Figure 4.16(a) to Figure 4.17(a). For example, 300 multi-terminal nets from Part 0 to (Part 1, Part 2) are split in 300 2-terminal nets from Part 0 to Part 1 and 300 2-terminal nets from Part 0 to Part 2. Then, the direction of the cut net is neglected in the modelling as shown from Figure 4.17(a) to Figure 4.17(b). For example, there are 1172 cut nets from Part 0 to Part 1 and 2938 cut nets from Part 1 to Part 0. In the modelling, they are counted as $1172+2938=4110$ cut nets between Part 0 and Part 1 without consideration of their directions.



(a) design after split (b) design model

Figure 4.17: *The modelling of the design*

4.8.1.2 Platform Generation

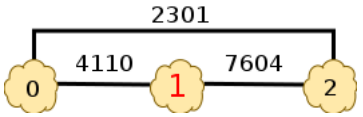
In order to generate the platform, the first step is to choose one part and then define the inter-FPGA tracks for its corresponding FPGA according to the distribution of cut nets. The critical path of multi-FPGA platforms is the inter-FPGA tracks where the cut nets pass through. Chapter 3 shows that the system clock frequency is limited by the maximum number of cut nets passing through one inter-FPGA track, thus limited by the highest ratio of the number of cut nets to the number of available I/Os. The part, which has the highest ratio of the number of cut nets to the number of available I/Os, has the highest priority to be chosen. When there are two parts having the same ratio that are the highest in all the parts, the part whose corresponding FPGA has less available I/Os is chosen. If the chosen part is Part X, the number of inter-FPGA tracks between FPGA X and another FPGA Y is proportional to the ratio of the number of cut nets between Part X and the Part Y to the total cut nets on Part X as shown in Equation 4.3. For example, the number of cut nets on each part after the modelling and the available I/Os per FPGA are shown in Figure 4.18(a). Part 1 is the chosen part (as shown in Figure 4.18(b)) due to that Part 1 has the highest ratio (as shown in Figure 4.18(a)). Therefore, inter-FPGA tracks of FPGA 1 will be distributed. Part 1 has 11714 total cut nets. Among the total cut nets, there are 4110 cut nets between Part 0 and Part 1 and 7604 cut nets between Part 1 and Part 2. Therefore, $1188 * \frac{4110}{11714} = 417$ tracks in case of Logic Multiplexing (resp. $567 * \frac{4110}{11714} = 199$ tracks in mode LVDS in case of ISERDES/OSERDES) will be distributed between FPGA 0 and FPGA 1, and $1188 * \frac{7604}{11714} = 771$ tracks in case of Logic Multiplexing (resp. $567 * \frac{7604}{11714} = 368$ tracks in mode LVDS in case of ISERDES/OSERDES) will be distributed between FPGA 1 and FPGA 2 as shown in Figure 4.18(c) (resp. as shown in

Figure 4.18(d)).

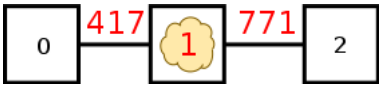
$$Tracks_{(X,\,Y)} = \frac{(Avail\ I/Os\ on\ X) * cut\ nets_{(X,\,Y)}}{cut\ nets\ on\ Part\ X} \tag{4.3}$$

Part	Cut Nets	FPGA	I/Os	ratio	LVDS	
					I/Os	ratio
Part 0	6411	FPGA 0	1045	7	497	13
Part 1	11714	FPGA 1	1188	10	567	21
Part 2	9905	FPGA 2	1189	9	569	18

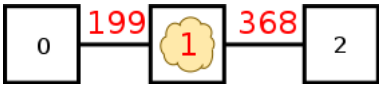
(a) choose the part



(b) design model



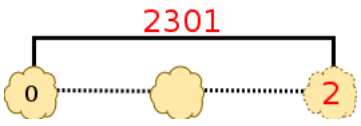
(c) the custom platform (I/Os)



(d) the custom platform (LVDS I/Os)

Figure 4.18: *The generation of the custom platform: Step 1*

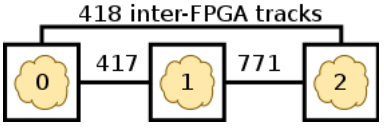
Once inter-FPGA tracks for an FPGA are defined, cut nets of the corresponding part will be removed from the design model. Then, the previous step is iterated to choose one part and define the corresponding inter-FPGA tracks. For example, after having defined the inter-FPGA tracks of FPGA 1, cut nets of Part 1 will be removed from the design model as shown in Figure 4.19(a). Then, the number of cut nets on each part and available I/Os for corresponding FPGAs are updated as shown in Figure 4.19(b). In the next iteration, Part 0 and Part 2 have the same number of cut nets as there are only two parts in the design modelling. Part 2 has less available I/Os, thus higher ratio of the number of cut nets to the number of available I/Os than Part 0. Therefore, inter-FPGA tracks of FPGA 2 will be defined. Part 2 has 2301 cut nets and all the cut nets on Part 2 are cut nets between Part 0 and Part 2. Therefore, 418 tracks in case of Logic Multiplexing (resp. 201 tracks in LVDS in case of ISERDES/OSERDES) will be defined between FPGA 0 and FPGA 2 as shown in Figure 4.19(c) (resp. as shown in Figure 4.19(d)).



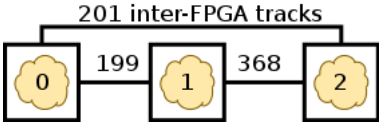
(a) remove the defined part

Part	Cut Nets	FPGA	I/Os	ratio	LVDS	
					I/Os	ratio
Part 0	2301	FPGA 0	628	4	298	8
Part 1	0	FPGA 1	0	---	0	---
Part 2	2301	FPGA 2	418	6	201	12

(b) update the cut nets and available I/Os



(c) the custom platform (I/Os)



(d) the custom platform (LVDS I/Os)

Figure 4.19: *The generation of the custom platform: Step 2*

After several iterations, all the inter-FPGA tracks are defined and the two platforms are generated. For example, after having defined the inter-FPGA tracks of FPGA 2, cut nets of Part 2 will be removed from the design model. After that, there is no cut net in the design model and all the inter-

FPGA tracks have been defined. According to Chapter 2, if ISERDES/OSERDES is chosen, one pair of FPGA I/Os per bank in mode LVDS between FPGAs is reserved for propagating the clock. In order that the inter-FPGA communication is full-duplex (one cut net can be propagated from FPGA A to FPGA B while another cut net from FPGA B to FPGA A), two pairs of FPGA I/Os per bank in mode LVDS between FPGAs are reserved. Therefore, the generated custom platform is shown in Figure 4.19(c) for Logic Multiplexing and in Figure 4.20 for ISERDES/OSERDES.

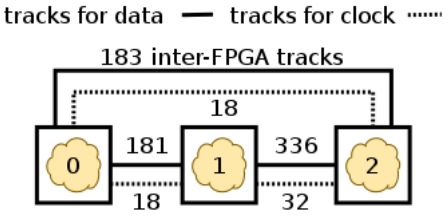
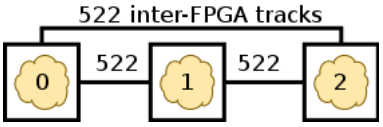


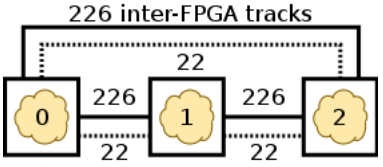
Figure 4.20: *the custom platform in ISERDES/OSERDES*

4.8.1.3 Comparison with the generic platform

After the platform generation, the generated custom platform will be compared with the generic platform. The generic platform has the same number of inter-FPGA tracks in each pair of FPGAs and has a DDR connected with each FPGA (as shown in Figure 4.21(a) for Logic Multiplexing and in Figure 4.21(b) for ISERDES/OSERDES). The global signals, the reset chain and the external interfaces are taken into consideration but not shown in the figures. In the custom platform, inter-FPGA tracks are distributed according to the distribution of cut nets. For example, there are more cut nets between Part 1 and Part 2. Therefore, there are more inter-FPGA tracks distributed between FPGA 1 and FPGA 2. In the example, the design has only 1 DDR and connected with the logic in Part 0. Therefore, only FPGA 0 is connected with 1 DDR.



(a) the generic platform (I/Os)



(b) the generic platform (LVDS I/Os)

Custom (I/Os): 10 cut nets per track
Generic (I/Os): 15 cut nets per track

(c) the mux

Custom: 23 cut nets per track
Generic: 34 cut nets per track

(d) the mux (LVDS I/Os)

Figure 4.21: *Comparison with the generic platform*

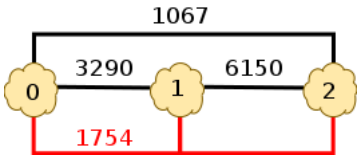
Comparing to the generic platform, due to the proposed interconnect synthesis algorithm, the maximum number of cut nets passing through one multiplexer (noted as mux) in the custom platform is reduced (from 15 to 10 as shown in Figure 4.21(c) for Logic Multiplexing and from 34 to 23 as shown in Figure 4.21(d) for ISERDES/OSERDES). The time-division-multiplexing

as shown in Chapter 2 need to be implemented in order to pass multiple cut nets through one multiplexer. In the time-division-multiplexing, less mux means higher performance.

4.8.2 Generation of the Custom Platform with 2- and Multi-Point Tracks

4.8.2.1 Design Modelling

The modelling keeps the multi-terminal nets but neglects the direction of all the cut nets for the generation of the platform with both 2- and multi-point tracks. The number of terminals from which the cut nets are counted for generating the multi-point tracks is noted as ter, which is an input parameter of the proposed automatic design flow. If the number of terminals is noted as n and the number of FPGAs in the platform is noted as m, according to Chapter 3, routing and multiplexing a multi-terminal net in a multi-point track can spare FPGA I/Os when $m \leq 2(n-1)$ and can waste FPGA I/Os when $m > 2(n-1)$. Thus, the input parameter ter need to fulfil $m \leq 2(ter-1)$. For example, the design in Figure 4.16(a) is modelled as shown in Figure 4.22(a). The number of FPGAs in the platform is 3. Therefore, the parameter ter is set to 3, meaning that cut nets from 3-terminal are counted for generating the multi-point tracks. There are 12261 cut nets in total and 1754 cut nets ($n \geq ter$) as shown in Figure 4.22(b).



(a) design model

Inter-FPGA nets	Num.
Nets (n >= ter)	1754
Total nets	12261

(b) cut nets after modelling

Figure 4.22: *The modelling of the design*

4.8.2.2 Percentage of FPGA I/Os for multi-point tracks

In this manuscript, only cut nets which have the same driver FPGA and the same receiver FPGAs are assumed to share a multi-point track due to the physical limitation and limitation of multiplexing techniques. The percentage of FPGA I/Os used for multi-point tracks is equal to the percentage of the cut nets ($\geq ter$) in the total cut nets. Consequently, the number of FPGA I/Os used for multi-point tracks (defined as shown in Equation 4.4) is the available FPGA I/Os multiplied by this percentage. Due to that several FPGAs are connected with external interfaces, different FPGAs can have different available FPGA I/Os. The available FPGA I/Os used in Equation 4.4 are the minimum number of them. After defining multi-point tracks, the $cut\ nets_{\geq ter)}$ are removed from the design model and available I/Os for each FPGA are updated. For example, the percentage of $cut\ nets_{\geq ter}$ is $\frac{1754}{12261} = 14.31\%$ as shown in Figure 4.22(a). After assigning the external interfaces, available FPGA I/Os are 1045 in case of Logic Multiplexing (resp. 497 in mode LVDS in case of ISERDES/OSERDES) as shown in Figure 4.23(a). Therefore, $1045 * \frac{1754}{12261} = 150$ in case

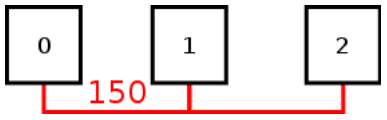
of Logic Multiplexing (resp. $497 * \frac{1754}{12261} = 72$ multi-point tracks in mode LVDS in case of ISERDES/OSERDES) are distributed as shown in Figure 4.23(b) (resp. as shown in Figure 4.23(c)).

$$Tracks_{(multi-point)} = \frac{cut\ nets_{n \geq ter)} * FPGA\ I/Os}{all\ the\ cut\ nets} \tag{4.4}$$

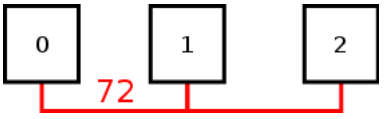
Then, the defined cut nets are removed from the design model as shown in Figure 4.23(e). The number of cut nets on each part and available I/Os for corresponding FPGAs are updated as shown in Figure 4.23(f). In order that multi-point tracks are multitasking, when there are m FPGAs in the platform, m pairs of FPGA I/Os per bank in mode LVDS in case of ISERDES/OSERDES are reserved for propagating the clock as shown in Figure 4.23(d).

FPGA	I/Os	LVDS I/Os
FPGA 0	1045	497
FPGA 1	1188	567
FPGA 2	1189	569

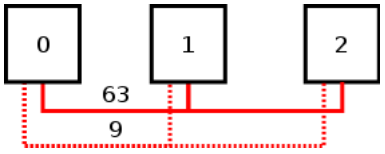
(a) available I/Os



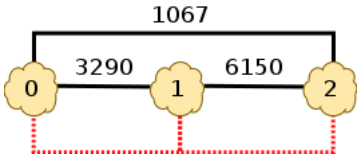
(b) the custom platform (I/Os)



(c) the custom platform (LVDS I/Os)



(d) reserve clock for multi-point



(e) remove defined cut nets

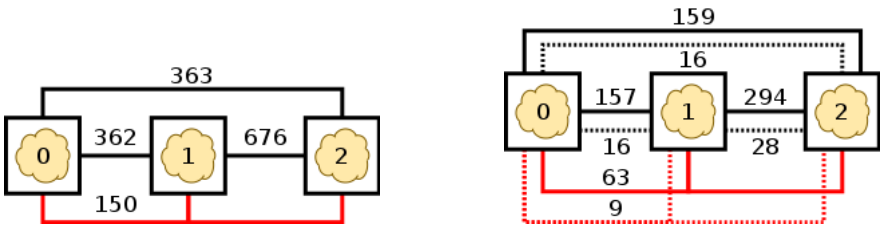
Part	Cut Nets	FPGA	I/Os	ratio	LVDS	
					I/Os	ratio
Part 0	4357	FPGA 0	895	5	425	11
Part 1	9440	FPGA 1	1038	10	495	20
Part 2	7217	FPGA 2	1039	7	497	15

(f) update the cut nets and available I/Os

Figure 4.23: *Generation of multi-point tracks*

4.8.2.3 Platform Generation

The rest of the design model will be processed according to the previous discussion to generate 2-point tracks. Finally, the generated custom platform is shown in Figure 4.24(a) for Logic Multiplexing (resp. as shown in Figure 4.24(b) for ISERDES/OSERDES).



(a) the custom platform (I/Os) (b) the custom platform (LVDS I/Os)

Figure 4.24: *result of the platform*

4.8.3 Conclusion

In this Section, we proposed the interconnect synthesis algorithm that distributes inter-FPGA tracks according to the distribution of cut nets in order to tailor the platform for the given design. The proposed interconnect synthesis algorithm is realized in python programming language. The results show that the generated platform can achieve less maximum number of cut nets passing through one multiplexer than the generic platform, thus higher performance. There are two types of inter-FPGA tracks: 2-point track and multi-point track. Therefore, the generated platforms have two types: "Platform with only 2-point tracks" and "Platform with 2- and multi-point tracks". In the experiments of Section 4.11, these two types of platforms will be compared in performance.

4.9 Automatic Design Flow for Creating a Custom Platform

The previous Sections have detailed the point tools for the main steps when creating a custom platform. In this Section, we propose an automatic design flow as depicted in Figure 4.25, which automatically joins the point tools. The proposed automatic design flow that is realized in python programming language, consists of the platform generation flow and the implementation flow. In the platform generation flow, a custom platform tailored for a given design is automatically generated by only choosing the FPGA type used. In the implementation flow, the partitioned design is routed into the generated multi-FPGA platform. As most of off-the-shelf platforms, there is only one FPGA type used in one custom platform. Therefore, different custom platforms can be identified by the input design and the FPGA type.

The automatic design flow starts by the platform generation flow. The inputs of the platform generation flow are the prototyped design RTL, the FPGA Lib and the user definition file. Through several steps, the flow generates the board netlist. The board netlist can be high-performance with considering the constraints for high-speed signaling (i.e. LVDS), high performance multiplexing (i.e. ISERDES/OSERDES) and so forth. The FPGA lib contains the logic capacity information and the user I/O information of different FPGA types. The user definition file contains the information of the chosen FPGA type, the maximum FPGA logic capacity utilization, the chosen inter-FPGA track type ("only 2 point tracks" or "2- and multi-point tracks"), and the chosen multi-

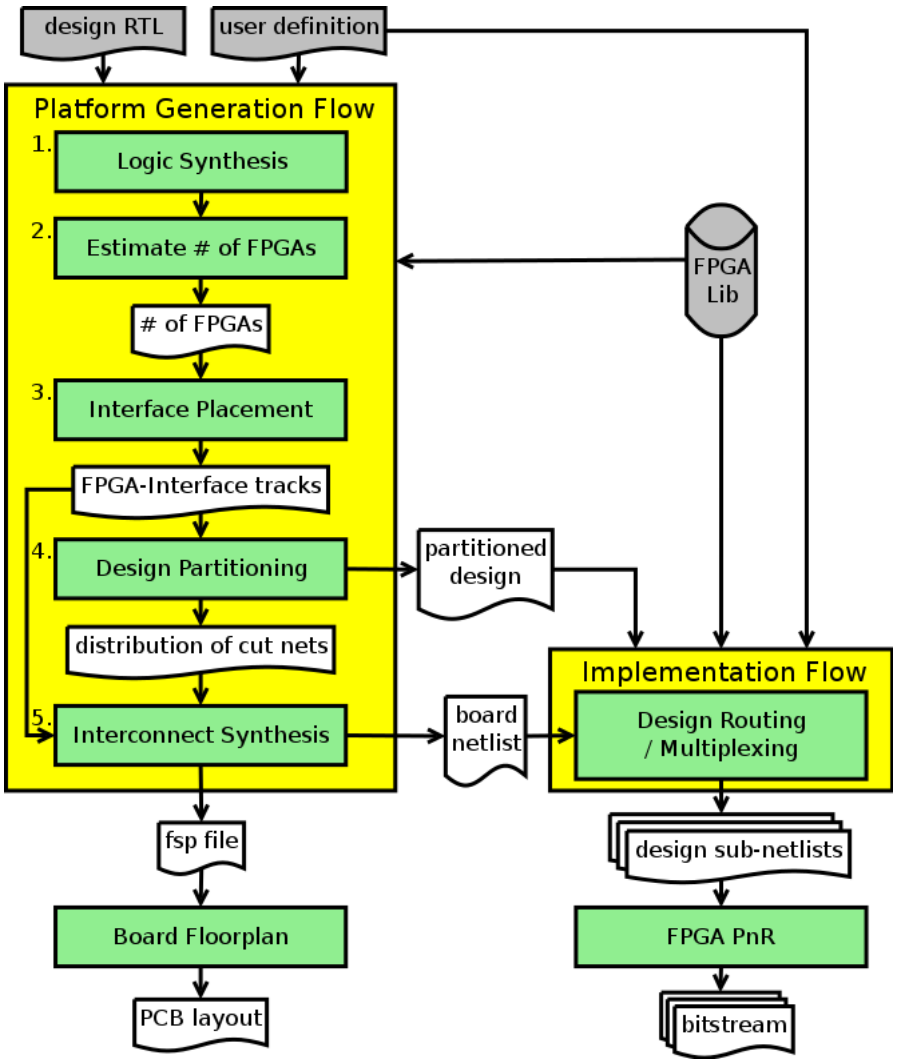


Figure 4.25: *The automatic design flow for creating a custom platform*

plexing architecture (Logic Multiplexing or ISERDES/OSERDES). Different steps of the platform generation flow are presented as follows.

1. Logic Synthesis: The input design is synthesized targeting FPGAs (from RTL to design netlist). This step can be realized by Xilinx XST Synthesis tool [XST, 2014], Altera Quartus Synthesis tool [Quartus, 2014], or third party tools such as Synopsys Synplify tool [Synplify, 2014] and Mentor Graphics Precision tool [Precision, 2014].

2. Estimate the number of FPGAs: When the logic elements in the prototyped design netlist exceeds the logic capacity of the chosen FPGA multiplied by the maximum FPGA logic capacity utilization, the multi-FPGA platform is needed. In this case, the minimum number of FPGAs required in the platform need to be estimated. This step can be realized by the commercial tool waCore provided by the company Flexras Technologies [Flexras, 2014]. As a result, an XML file that contains the information of the number of FPGAs required in

the platform is generated.

3. Interface Placement: External interfaces to FPGAs are connected. We propose an External Interface Placement Algorithm in Section 4.6 to do this step. The proposed external interface placement algorithm is realized in python programming language. At the end of this step, the FPGA-to-Interface tracks file in XML format is generated.

4. Design Partitioning: As the SoC/ASIC design is bigger than the FPGA, the design netlist is partitioned into several parts according to the number of FPGAs. Each part's capacity fits in a single FPGA. This step is done by the commercial tool waPart provided by the company Flexras Technologies [Flexras, 2014]. The partitioning needs to take the connection constraints (the distribution of inter-FPGA connections) into consideration, in order to achieve a higher performance solution. We propose an algorithm to generate a temporary balanced platform in Section 4.7 (realized in python programming language). At the end of this step, the partitioned design netlist in Verilog format and the cut nets distribution file in XML format are generated.

5. Interconnect Synthesis: Inter-FPGA tracks are distributed according to the distribution of cut nets. We propose an Interconnect Synthesis Algorithm in Section 4.8 to do this step. The proposed interconnect synthesis algorithm is realized in python programming language. Finally, the board netlist in Verilog format and the fsp format file (fsp: FPGA System Planner) are generated. The fsp format file can be loaded by Cadence Board Floorplan tool Allegro [FSP, 2014] to generate PCB layout. The screenshot of an example generated by the proposed automatic design flow is depicted in Figure 4.26.

After finishing the platform generation flow, the automatic design flow passes to the implementation flow. The implementation flow takes the partitioned design netlist, the board netlist, the FPGA lib and the user definition file as inputs. Due to that the design has already been synthesized and partitioned in the platform generation flow, the implementation is to route and multiplex the partitioned design into the generated platform to verify the functionality and to evaluate the performance. We propose a routing algorithm (that enhances the Turki's algorithm [Turki et al., 2013]) in Chapter 3 to do the design routing. The proposed routing algorithm is realized in C++ programming language. It generates a TCL format file that contains the information of the cut net to inter-FPGA track mapping. Then, the commercial tool waRoute provided by the company Flexras Technologies [Flexras, 2014] will read this TCL format file, in order to multiplex several cut nets onto a single track and add the multiplexing IPs in each part of the partitioned design. Finally, the design sub-netlists in Verilog format are generated, which can be loaded by FPGA PnR tools to generate the bitstreams.

The proposed automatic design flow optimizes the cost and the performance while reducing the time of availability. In the following, the evaluations of the cost, the performance and the time of availability are discussed.

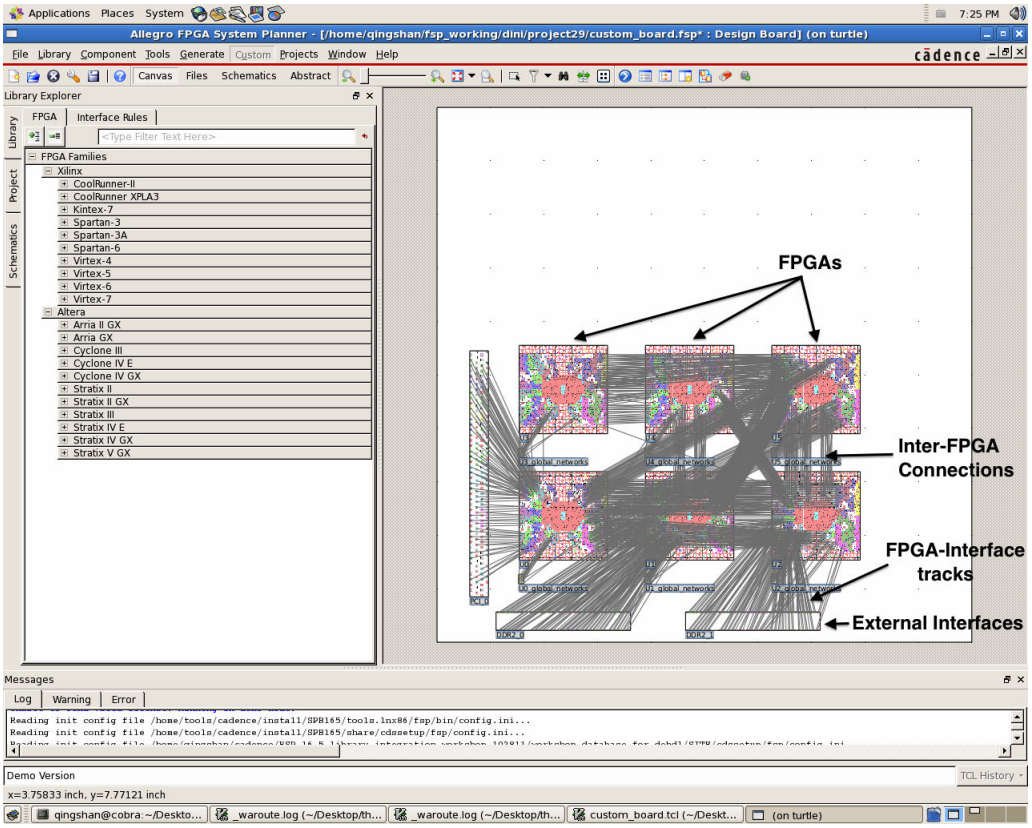


Figure 4.26: *The generated results loaded in the Floorplan tool Allegro*

4.9.1 Cost Evaluation

According to [Amos et al., 2011], the cost of creating a custom platform consists of the personnel cost, the material cost, the assembly cost, and the opportunity cost. An example of a custom platform with 4 Virtex-6 FPGAs is studied in [Amos et al., 2011]. The costs of the example are depicted in Table 4.2 and detailed in the following.

Table 4.2: Main costs associated with custom platform development

Costs	Min	Max
Personnel cost	$10,000	$15,500
Material cost (per board)	$50,357	$58,333
Assembly cost (per board)	$3,000	$4,333
Opportunity cost	$37,500	$58,245
Total cost	$47,500 + ($53,357/board)	$73,745 + ($62,666/board)

- Personnel cost: It consists of the costs of the FPGA circuit development (from design RTL to board netlist), the PCB layout designer, and the test engineer. The manufacture of PCB is assumed to be out-sourced.

- Material cost: After the custom platform has been designed, it has to be built and this requires Bill of Materials (BoM) management and component purchase effort. The approximate cost for the material includes the PCB manufacture, the board itself, the FPGAs, and other components. Nearly half of the material cost would be the cost of FPGAs [Amos et al., 2011].
- Assembly cost: Assembly can be done in-house or be out-sourced.
- Opportunity cost: It is expressed as a missed Return on Investment (ROI), meaning that benefit which has not been realized elsewhere because resources are allocated to the given project.

If more copies of the custom platform are needed for remote deployment, the personnel cost and the opportunity cost will not be changed and can be neglected (i.e. if more than 10 copies are needed, the personnel cost and the opportunity cost can be neglected). Therefore, the rest costs are the material cost and the assembly cost. The material cost is ten times more than the assembly cost. Thus, the assembly cost can be neglected. Nearly half of the material cost would be the cost of FPGAs [Amos et al., 2011]. Therefore, in this manuscript, the cost evaluation can be defined as Equation 4.5. The number of copies (boards) of the custom platform is noted as NUM_{Board}. The cost of FPGAs is the multiple of the price of one chosen FPGA (noted as $Price_{FPGA}$) and the number of FPGAs required in the custom platform (noted as NUM_{FPGA}).

$$\begin{aligned} Cost &= Material\ cost * NUM_{Board} \\ &= 2 * Price_{FPGA} * NUM_{FPGA} * NUM_{Board} \end{aligned} \tag{4.5}$$

The prices of different FPGAs ($Price_{FPGA}$) are stored in FPGA Lib detailed in Section 4.3.1, which is established before launching the automatic design flow. After choosing one FPGA and launching the automatic design flow, the flow estimates the number of FPGAs required in the custom platform (NUM_{FPGA}) according to Section 4.5. Finally, the cost of one custom platform copy will be obtained according to Equation 4.5.

4.9.2 Performance Evaluation

As discussed in Section 2.3, the achieved performance of multi-FPGA platforms is influenced by the inter-FPGA communication. There are two different inter-FPGA communication architecture: Logic Multiplexing and ISERDES/OSERDES. The maximum number of cut nets passing through one multiplexer (logic multiplexer in Logic Multiplexing, serdes in ISERDES/OSERDES) is noted as mux. According to Chapter 3, intermediate FPGAs may be necessary if there is no available track between the FPGA (the Driver) and the FPGA (the Receiver) in Design Routing process. The maximum number of intermediate FPGAs needed (routing hop) is noted as hop. When implementing Logic Multiplexing (resp. ISERDES/OSERDES), the performance (in terms of the

system clock frequency) can be defined as Equation 4.6 (resp. as Equation 4.7) as detailed in Section 3.6.3.

$$sys\_clk = \frac{if\_clk}{lat} = \frac{125}{mux + hop + 3} MHz \tag{4.6}$$

$$sys\_clk = \frac{if\_clk}{lat} = \frac{250\ MHz}{(7 + ceil(\frac{mux}{4})) * (1 + hop)} \tag{4.7}$$

4.9.3 Time of Availability Evaluation

The time of availability is an essential element for FPGA-based prototyping. The earlier the prototyping platform is available, the earlier the developers can start the hardware/software integration. Therefore, the time spent for creating a custom platform need to be taken into consideration. An example of a custom platform with 4 Virtex-6 FPGAs is studied in [Amos et al., 2011]. The time spent of the example is depicted in Table 4.3 and can be different according to different experts. When creating the custom platform for a specific design, there are different tasks from design RTL to final PCB board. These tasks include: FPGA circuit development (from design RTL to board netlist), PCB layout, PCB manufacture, Assembly, Test and Rework. PCB manufacture and assembly are assumed to be out-sourced.

Table 4.3: Time spent for typical tasks of custom platforms

Task	Min	Max
FPGA circuit development (from design RTL to board netlist)	1 month	2 month
PCB layout	2 month	2.5 month
PCB manufacture	0.5 month	1 month
Assembly (per board)	0.15 month + (0.025 month/board)	0.25 month + (0.025 month/board)
Test (per board)	0.05 month/board	0.1 month/board
Rework	0 month	1 month
Total time spent	3.65 month + (0.075 month/board)	6.75 month + (0.125 month/board)

Table 4.3 shows that more copies (boards) of the custom platform do not add many more overheads in time. The proposed design flow automatizes the FPGA circuit development. Therefore, with the help of the proposed automatic design flow, the time spent of the FPGA circuit development reduces from 1-2 months to several hours, which can be neglected compared to other tasks shown in Table 4.3.

4.9.4 Conclusion

In this Section, we proposed the automatic design flow for creating a custom platform. The cost evaluation, the performance evaluation and the time of availability evaluation are discussed. The advantages of the proposed automatic design flow are to optimize the cost and the performance while reducing the time-to-market and lowering the barrier of the board designers.

4.10 Board Exploration

There are many different types of FPGAs (i.e. vendor: Xilinx [Xilinx, 2014] or Altera [Altera, 2014], family: Virtex-7 or Stratix5, device: 2000T or GXAB, and package: FLG1925 or F1932). Different FPGA types have different logic capacity, different numbers of FPGA I/Os and different prices. If different FPGAs are chosen, the generated multi-FPGA platforms have different costs and different performances. The board exploration is defined to generate different multi-FPGA platforms by choosing different FPGA types in the FPGA circuit development (from design RTL to board netlist). Then, an optimum prototyping platform (board netlist) will be chosen according to designers' specifications. According to Section 4.9, the board exploration is not possible when the FPGA circuit development is manual and one trial needs 1-2 months. The proposed design flow, which automatizes the FPGA circuit development, permits the board exploration due to that each launch of the flow spends only several hours. An example of the board exploration is shown in Figure 4.27. Three different custom platforms are generated by the automatic design flow with three different FPGA types. These FPGA types have the same number of FPGA I/Os and different logic capacities. According to their logic capacities, these FPGAs are classified as small FPGAs, middle FPGAs and large FPGAs. Different custom platforms have different costs and achieve different performance. The optimum prototyping platform (cost-optimal, performance-optimal, or intermediate) can be chosen.

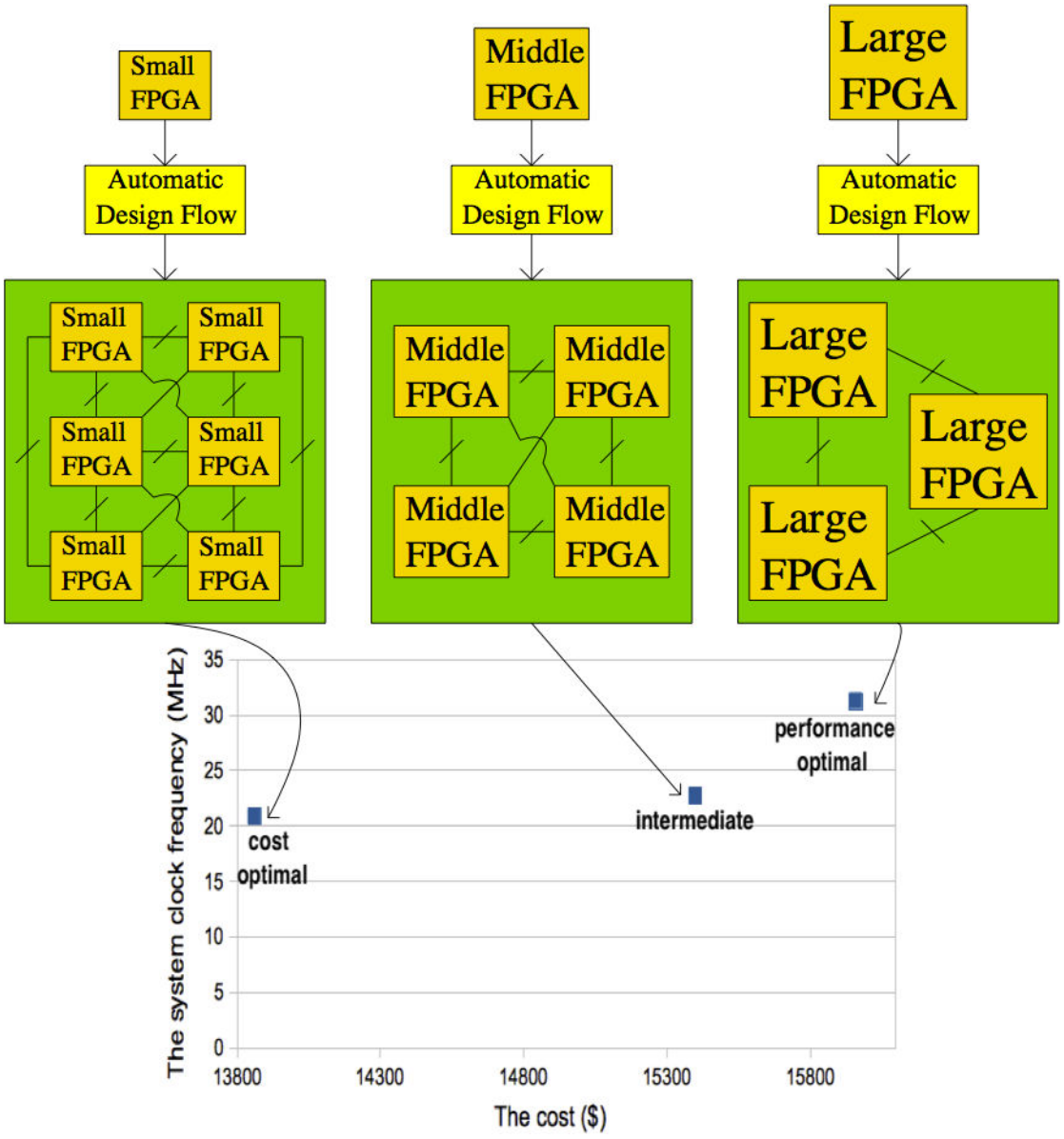


Figure 4.27: *Board exploration*

4.11 Experiments

Experiments are conducted using four testbenches (leon2, leon3_avnet, leon3mp, netcard) from Gaisler Research Benchmarks [Gaisler, 2014] and three testbenches (vga_lcd, des_perf, ethernet) from OpenCores Benchmarks [OpenCores, 2014]. According to our knowledge, the Gaisler Research Benchmarks are the largest available benches representative of industrial designs. The chosen OpenCores Benchmarks have a high ratio between multi-terminal nets and total cut nets that permits to test the utility of the multi-point tracks. The performance (in terms of the system clock frequency) is noted as sys_clk. The logic capacity utilization rate of each testbench

is the same with the experiments in Chapter 3. The proposed automatic design flow has different scenes of custom platforms due to different inter-FPGA communication architectures (Logic Multiplexing and ISERDES/OSERDES) and different types of inter-FPGA tracks ("only 2-point tracks" and "2- and multi-point tracks"). First, custom platforms using Logic Multiplexing and ISERDES/OSERDES will be compared in performance. Second, custom platforms using ("only 2-point tracks" and "2- and multi-point tracks") will be compared in performance. Finally, the board exploration will be done.

4.11.1 Logic Multiplexing VS ISERDES/OSERDES

Custom platforms using Logic Multiplexing and ISERDES/OSERDES are compared as shown in Table 4.4. The chosen FPGA type is XC5VLX330FF1760 in order to compare with the targeted off-the-shelf platform used in Chapter 3. Therefore, the generated custom platforms have 6 FPGAs. The number of cut nets passing through one multiplexer is noted as mux ($mux = mux\_logic$ in Logic Multiplexing and $mux = mux\_serdes$ in ISERDES/OSERDES). The routing hop is noted as hop. The performance using Logic Multiplexing is calculated according to Equation 3.1 and using ISERDES/OSERDES is evaluated according to Equation 3.2. Logic Multiplexing can be noted as LM and the gain of ISERDES/OSERDES over Logic Multiplexing is calculated.

Table 4.4: Logic Multiplexing VS ISERDES/OSERDES

testbench	Logic Multiplexing			ISERDES/OSERDES			gain vs Logic Multiplexing
	mux	hop	sys_clk	mux	hop	sys_clk	
leon2	8	1	10.42 MHz	21	0	19.23 MHz	85%
leon3_avnet	4	1	15.63 MHz	20	0	20.83 MHz	33%
leon3mp	3	0	20.83 MHz	7	0	27.78 MHz	33%
netcard	15	0	6.94 MHz	44	0	13.89 MHz	100%
vga_lcd	5	0	15.63 MHz	16	0	22.73 MHz	45%
des_perf	3	0	20.83 MHz	4	0	31.25 MHz	50%
ethernet	3	0	20.83 MHz	6	0	27.78 MHz	50%

The results show that ISERDES/OSERDES can always achieve higher performance than Logic Multiplexing in the custom platform as depicted in Figure 4.28. The gain in performance is up to 100%. Nevertheless, according to Chapter 3, Logic Multiplexing can achieve higher performance than ISERDES/OSERDES with several testbenches in the off-the-shelf platform. This is due to that the proposed automatic design flow distributes the inter-FPGA tracks according to the distribution of cut nets permitting no routing hop in ISERDES/OSERDES, while the off-the-shelf platform has generic and balanced connections. In the following, the custom platform uses ISERDES/OSERDES as the inter-FPGA communication architecture.

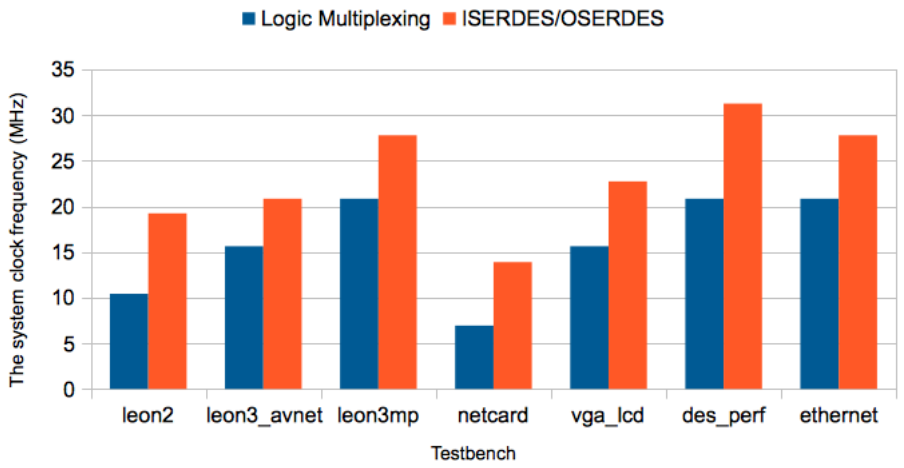


Figure 4.28: *Logic Multiplexing VS ISERDES/OSERDES*

4.11.2 "Platform with Only 2-Point Tracks" VS "Platform with 2- and Multi-Point Tracks"

Custom platforms using ("only 2-point tracks" and "2- and multi-point tracks") are compared in performance in this sub section. The number of terminals from which the cut nets are counted for generating the multi-point tracks is noted as ter, which is the input parameter of the proposed automatic design flow. If $ter \geq 4$, only multi-terminal nets that have no less than 4 terminals are counted for multi-point tracks. The number of terminals is noted as n and the number of FPGAs in the platform is noted as m. According to Chapter 3, routing and multiplexing a multi-terminal net in a multi-point track can spare FPGA I/Os when $m \leq 2(n-1)$ and can waste FPGA I/Os when $m > 2(n-1)$. As the chosen FPGA type is XC5VLX330FF1760 in order to compare with the targeted off-the-shelf platform used in Chapter 3, the generated custom platforms have 6 FPGAs. Therefore, the minimum value of ter is 4 in order to spare FPGA I/Os. The number of multi-point tracks in the generated custom platform is noted as num. For each parameter ter, the gain of "2- and multi-point tracks" platform over "only 2-point tracks" platform is calculated.

4.11.2.1 multi-point tracks are more critical than 2-point tracks in terms of performance

As discussed in Chapter 3, multi-point tracks are more critical than 2-point tracks in terms of performance in the off-the-shelf platform. With taking this constraint into consideration, the results of the performance comparison is depicted in Table 4.5. Only the testbenches have a high ratio between multi-terminal nets and total cut nets (as shown in Table 3.1) are chosen in the experiments.

Table 4.5: Scene 1: 2-point track VS 2- and multi-point track

testbench	2-point	2- & multi- point								
		$ter \geq 6$			$ter \geq 5$			$ter \geq 4$		
	sys_clk (MHz)	num	sys_clk (MHz)	gain %	num	sys_clk (MHz)	gain %	num	sys_clk (MHz)	gain %
netcard	13.89	0	13.89	0	220	11.36	-18	330	10.42	-25
vga_lcd	22.73	140	20.83	-8	240	20.83	-8	330	20.83	-8
des_perf	31.25	10	31.25	0	30	31.25	0	100	27.78	-11
ethernet	27.78	0	27.78	0	10	27.78	0	210	27.78	0

The results show that the "2- and multi-point tracks" platform achieves lower performance than the "only 2-point tracks" platform as depicted in Figure 4.29. For each testbench, the less is ter, the more is the num and the lower is the performance. The gain in performance is negative and up to -8% ($ter = 6$), -18% ($ter = 5$) and -25% ($ter = 4$). This is due to that the maximum number of cut nets passing through a multi-point track should be a half of that through a 2-point track, in order to avoid that the critical path is multi-point tracks and increase the performance.

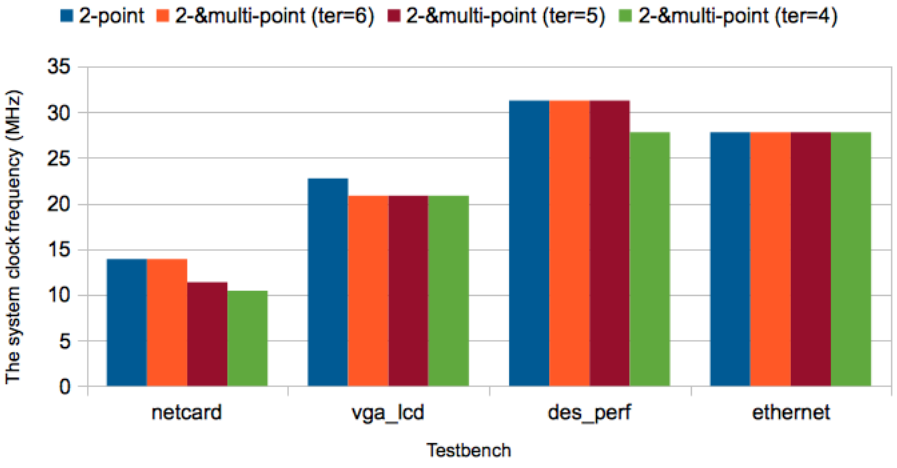


Figure 4.29: *Scene 1: 2-point tracks VS 2- and multi-point tracks*

4.11.2.2 multi-point tracks are equal with 2-point tracks in terms of performance

The reason of that multi-point tracks are more critical than 2-point tracks in terms of performance in the off-the-shelf platform, is that multi-point tracks have a longer trace than 2-point tracks in PCB board, thus a larger delay. Assume that PCB designers can increase the performance of multi-point tracks without bothering the performance of 2-point tracks by efficiently designing the custom platform PCB layout. In this case, the maximum number of cut nets passing through a multi-point track can be the same with that through a 2-point track. Custom platforms using ("only 2-point tracks" and "2- and multi-point tracks") will be compared in performance as shown

in Table 4.6.

Table 4.6: Scene 2: 2-point track VS 2- and multi-point track

testbench	2-point	2- & multi- point								
		$n \geq 6$			$n \geq 5$			$n \geq 4$		
	sys_clk (MHz)	num	sys_clk (MHz)	gain %	num	sys_clk (MHz)	gain %	num	sys_clk (MHz)	gain %
netcard	13.89	0	13.89	0	220	14.71	6	330	14.71	6
vga_lcd	22.73	140	22.73	0	240	25	10	330	25	10
des_perf	31.25	10	31.25	0	30	31.25	0	100	31.25	0
ethernet	27.78	0	27.78	0	10	27.78	0	210	31.25	12

The results shows that the "2- and multi-point tracks" platform achieves higher performance than the "only 2-point tracks" platform as depicted in Figure 4.30. For each testbench, the less is the ter, the more is the num and the higher is the performance. The gain in performance is positive and up to 0 ($ter = 6$), 10% ($ter = 5$) and 12% ($ter = 4$). Nevertheless, the performance gain is very limited and multi-point tracks increase the difficulty of PCB designs. Therefore, the custom platform using "only 2-point tracks" as inter-FPGA tracks is preferred. In the following, the custom platform is the platform using "only 2-point tracks".

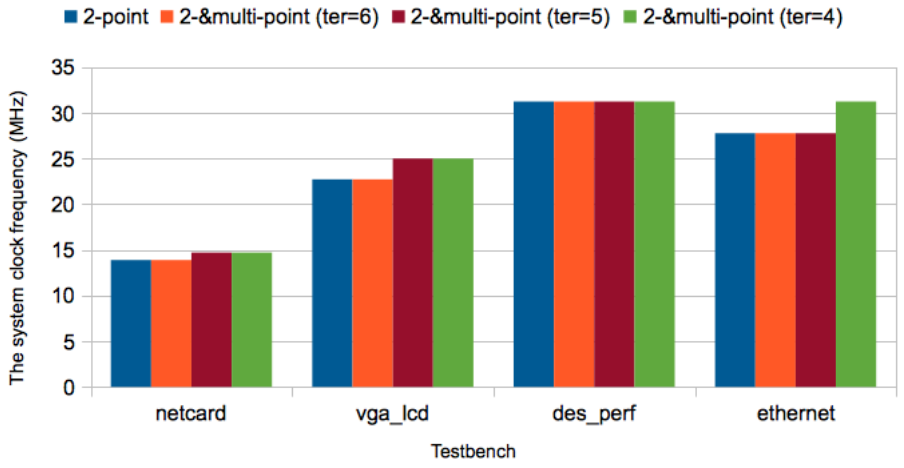


Figure 4.30: *Scene 2: 2-point tracks VS 2- and multi-point tracks*

4.11.3 Board Exploration

Gaisler Research Benchmarks [Gaisler, 2014] are chosen in the experiments for the board exploration, due to that they are the largest available benches representative of industrial designs. One advantage of the proposed automatic design flow is reducing the time to craft a custom platform. In order to study this advantage, the CPU time and the occupied internal memory size (noted as MEM size) are measured. Another advantage is to permit board designers to choose the optimum

FPGA type according to their specification (i.e. the cost and the performance). Three different FPGA types, which are widely used in the off-the-shelf prototyping platforms, are used in the experiments to generate the custom platforms. FPGA type 1 is XC5VLX330FF1760, FPGA type 2 is XC6VLX550TFF1759 and FPGA type 3 is XC6VLX760FF1760. The information of their logic capacity, I/O capacity and price is shown in Table 4.1. As most off-the-shelf platforms, all the FPGAs for implementing the ASIC logic have the same FPGA type in one generated custom platform. Therefore, the custom platform can be represented by the chosen FPGA type. In the experiments, each testbench is implemented respectively in three different custom platforms as shown in Table 4.7. These results are obtained by changing the parameters in the automatic design flow and relaunching the flow to get the achieved performances. The used PC has 4 intel xeon cores and maximum 16 GB internal memory. The filling rate is the maximum logic capacity utilization rate which ranges from 40% to 65%. The cost of the generated platform is calculated according to the Equation 4.5. For each generated custom platform, ten copies (boards) are needed. The performance is calculated according to the Equation 4.7 as ISERDES/OSERDES is taken as the multiplexing architecture. The number of external interfaces is noted as $\# \ of \ XIs$.

Table 4.7: Board Exploration

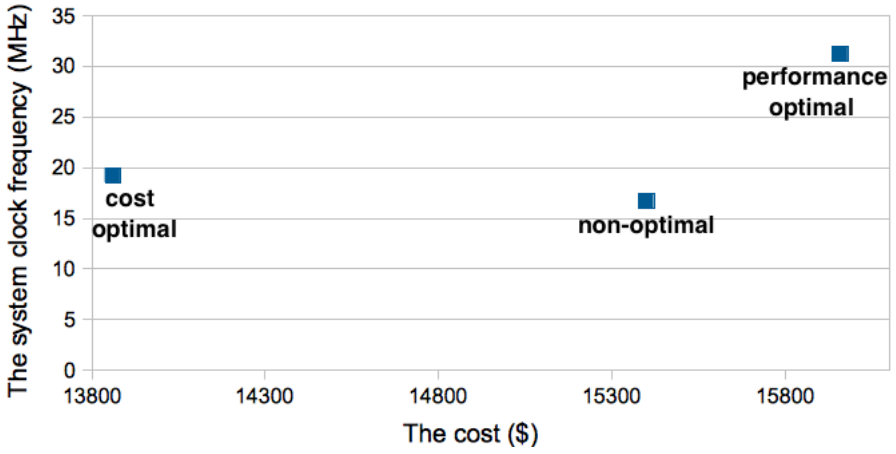
testbench	FPGA type	Filling rate (%)	# of XIs	CPU time (h)	MEM size (GB)	Price ($)	# of FPGA	Cost ($)	sys_clk (MHz)
leon2	type 1	60	2	2.8	4.2	2,310	6	13,860	19.23
	type 2	60	2	2.5	4.3	3,850	4	15,400	16.67
	type 3	60	2	2.5	4.2	5,320	3	15,960	31.25
leon3_avnet	type 1	65	10	12.3	4.7	2,310	6	13,860	20.83
	type 2	65	10	11.4	4.7	3,850	4	15,400	22.73
	type 3	65	10	10.9	4.7	5,320	3	15,960	31.25
leon3mp	type 1	40	7	6.4	2.9	2,310	6	13,860	27.78
	type 2	40	7	6.3	2.8	3,850	4	15,400	31.25
	type 3	40	7	6.1	2.7	5,320	3	15,960	31.25
netcard	type 1	55	3	5.5	3.5	2,310	6	13,860	13.89
	type 2	55	3	3.7	3.5	3,850	4	15,400	7.6
	type 3	55	3	3.4	3.5	5,320	3	15,960	20.83

According to Table 4.3, crafting a custom platform for a given design is a time-consuming process. There are seven different tasks when crafting the custom platform: FPGA circuit development (from design RTL to board netlist), PCB layout, PCB manufacture, assembly, test and rework. Among these steps, the FPGA circuit development takes about 1-2 months. Therefore, this is impossible to do the board exploration due to the cost of the time. The proposed automatic design flow permits the board exploration by totally automatizing the FPGA circuit development process. The results show that one launch takes several hours (2-12 hours) with an acceptable

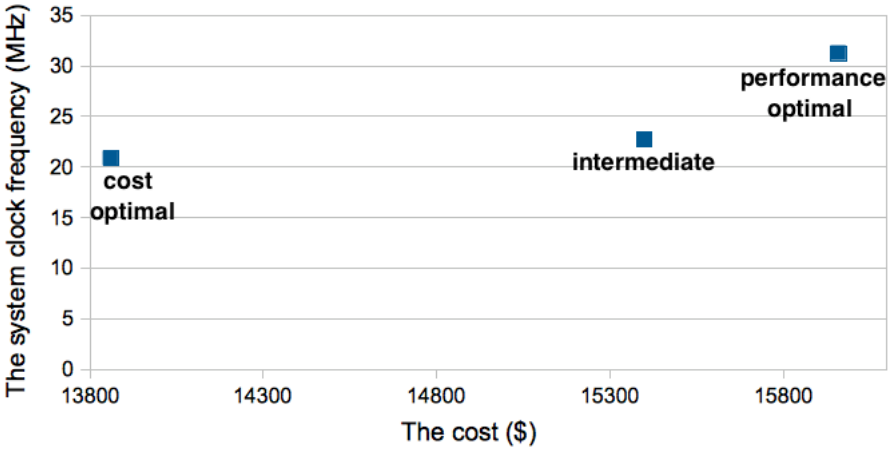
internal memory occupied (about 2-4 GB). In case of the same FPGA type and the same number of FPGAs in the platform, the CPU time is longer when there are more external interfaces in the testbench. In case of the same testbench and the same filling rate, the CPU time is longer when there are more FPGAs in the platform. If the testbench has larger logic capacity (logic capacity of testbenches is classified as follows: $leon3\_avnet > leon2 > netcard > leon3mp$), larger MEM size is occupied.

For the same testbench, different custom multi-FPGA platforms have different costs and different performance. An optimal solution means that no one (the cost or the performance) can be made better off without making another worse off. There are three different optimal solutions: cost-optimal, performance-optimal and intermediate. Board designers can choose the most adapted multi-FPGA board according to their specifications in the optimal solutions for the given design. For example, when leon2 is chosen, the solutions with the FPGA type (type 1 and type 3) are optimal solutions as shown in Figure 4.31(a). The FPGA type 1 solution achieves the lowest cost and the FPGA type 3 solution achieves the highest performance. Nevertheless, the FPGA type 2 solution is neglected due to that another solution with the FPGA type 1 can achieve the higher performance with lower cost. When leon3_avnet is chosen, all the solutions are optimal solutions as shown in Figure 4.31(b). The solution with the FPGA type 2 is an intermediate solution due to that it can achieve higher performance but higher cost than the FPGA type 1 solution, and lower performance but lower cost than the FPGA type 3 solution.

The logic capacity of the chosen FPGAs is classified as follows: FPGA type 1 $<$ FPGA type 2 $<$ FPGA type 3. In most cases, a larger FPGA that has higher logic capacity and higher price can achieve higher performance than a smaller FPGA that has lower logic capacity and lower cost. It is because there is less FPGAs in the multi-FPGA platform with a larger FPGA and less (mux and hop) can be achieved. Nevertheless, in several cases, a smaller FPGA that is less expensive could be used to reduce the overall cost of the custom platform, with higher or equal performance than a larger FPGA. For example, the solution with FPGA type 1 (resp. FPGA type 2) that is less expensive can achieve higher (resp. equal) performance than the solution with FPGA type 2 (resp. FPGA type 3) in the testbenches leon2 and netcard (resp. leon3mp).



(a) leon2



(b) leon3_avnet

Figure 4.31: *Board exploration (optimal solutions)*

4.12 Conclusion

In this Chapter, the custom platform is discussed. Nevertheless, crafting a home-made custom multi-FPGA platform is today a manual process, thus is time-consuming. The performance and the cost of the platform lie on the FPGA expertise and SoC DUT knowledge of the prototyping team. The board exploration with different FPGA types, which helps the engineers to design an optimum prototyping platform, can not be done. As the ratio between the logic capacity and the number of FPGA I/Os is increasing at an exponential rate, it becomes more and more challenging to design a high-performance custom multi-FPGA platform. The main contribution of this Chapter is to propose an automatic design flow for creating a custom platform, thus increasing the productivity, enabling board exploration, and optimizing cost and performance. The proposed automatic design flow automatically joins the point tools. We have developed several point tools such as the external interface placement tool that automatically find a solution for the external interface placement, the interconnect synthesis tool that automatically distribute inter-FPGA tracks accord-

ing to the distribution of cut nets. Experiments are conducted using four testbenches from Gaisler Research Benchmarks and three testbenches from OpenCores Benchmarks. The results show that even though Logic Multiplexing can achieve higher performance than ISERDES/OSERDES with several testbenches in the off-the-shelf platform, ISERDES/OSERDES can always achieve higher performance than Logic Multiplexing in the custom platform with up to 100% performance gain. Even though many off-the-shelf platforms are the platforms using "2- and multi-point tracks" as inter-FPGA tracks, the custom platform using "only 2-point tracks" as inter-FPGA tracks is preferred (multi-point tracks only used for global signals as presented in Section 4.8). With the proposed automatic design flow, the custom platform can be generated with an acceptable CPU time (several hours) and an acceptable internal memory resource (several GB). The board exploration can be done to permit that designers can choose the most adapted multi-FPGA board according to their specifications. The board exploration also shows that the smaller FPGA that is less expensive could be used to reduce the overall cost of the platform with sometimes higher performance.

The next Chapter focuses on the cabling platform. We propose a cabling platform with an algorithm to automatically optimize the cable distribution and the external interface placement.

Chapter 5

Cabling Multi-FPGA Platform

5.1 Introduction

The previous Chapter has discussed the custom platform and proposed the automatic design flow for creating a custom platform. The purpose of this Chapter is to discuss the cabling platform and propose an automatic design flow for creating a cabling platform. The main contribution is threefold. First, an algorithm is proposed to optimize the distribution of the cables. Then, with the help of the automatic tool, the optimal width of connectors for a cabling platform is explored. Finally, we propose a cabling platform where one board is composed of one FPGA and several connectors. All the FPGA I/Os are used for FPGA-to-connector connections. The connections inter FPGAs as well as the connections to external interfaces can be added or removed by only connecting or disconnecting the cables (resp. daughter boards) with or from the connectors.

The rest of this Chapter is organized as follows. Section 5.2 explains the overview of the cabling platform. Section 5.3 presents the proposed automatic design flow, more specifically an algorithm optimizing the distribution of the cables. In Section 5.4, the optimal width of connectors for a cabling platform is explored. Then, Section 5.5 proposes the cabling platform. Finally, Section 5.6 concludes this Chapter.

5.2 Platform Overview

The cabling platform, which is a relatively new notion compared to the off-the-shelf platform and the custom platform, consists of multiple ready-made FPGA boards connected by cables and connectors. In between the off-the-shelf and the custom platform, the cabling platform is semi off-the-shelf as it consists of multiple ready-made boards, and semi custom as its connections inter FPGAs as well as connections to external interfaces are user-defined and tailored for a specific design. Nevertheless, all the inter-FPGA connections are realized using cables and connectors instead of PCB traces.

As discussed in Section 2.2.3 of the state of the art, there are the two existing cabling platforms (proFPGA and Synopsys HAPS). The proFPGA cabling platform is proposed by [Prodesign, 2014]. For each FPGA, there are only 4 connectors. Among the four connectors, three of them have 148 pairs of tracks, and the rest one has 98 pairs of tracks. Nevertheless, the coarse-grained connector may waste FPGA I/Os when external interfaces do not occupy so many FPGA I/Os but need at least one connector. The HAPS cabling platform is proposed by Synopsys [HAPS, 2014]. For each FPGA, there are 23 connectors. Each connector has 24 pairs of tracks. Comparing to proFPGA, the connector in HAPS is fine-grained. For example, the external interface DDR3 that occupies 72 pairs of tracks, covers 3 connectors in HAPS. Therefore, no FPGA I/O is wasted. In proFPGA, this DDR3 covers the connector with 98 pairs of tracks and 26 pairs of FPGA I/Os are wasted.

As discussed in Section 2.3, there are two different inter-FPGA communication architectures used for FPGA-based prototyping: Logic Multiplexing and ISERDES/OSERDES. As the distribution of the inter-FPGA tracks in the custom platform, the distribution of the cables in the cabling platform is tailored for the given design. According to Chapter 4, with $125Mbps$ data rate, Logic Multiplexing achieves lower performance than ISERDES/OSERDES when the platform is customized for the design. Moreover, due to that the inter-FPGA delay and the tolerance delay in the cabling platform is higher than in the hardwired platform, the data rate is heavily limited in the cabling platform when implementing Logic Multiplexing that is system-synchronous. Nevertheless, the data rate in ISERDES/OSERDES (that is source-synchronous and propagates the clock along with the data), is not influenced by these limitations. Therefore, we will focus on ISERDES/OSERDES and discuss the constraints of ISERDES/OSERDES in the inter-FPGA communication of the cabling platform.

As the ISERDES/OSERDES is implemented in the cabling platform, each inter-FPGA track consumes a pair of FPGA I/O pins in LVDS. The width of connectors (resp. the cables) is the number of inter-FPGA tracks in mode LVDS connected to this connector, and is assumed to be the same for all the connectors (resp. all the cables). For example, there are 12 inter-FPGA connections (6 tracks in mode LVDS) between one connector and one FPGA as shown in Figure 5.1, therefore the width of connectors is 6. The multiple of the width of connectors and the maximum number of the connectors to one FPGA should be less than the maximum number of FPGA I/O pairs in mode LVDS as shown in Equation 5.1. For example, if the chosen FPGA in the cabling platform is XC7V2000TFLG1925 (vendor: Xilinx, family: Virtex-7, device: 2000T, and package: FLG1925, thus FPGA type: XC7V2000TFLG1925), the maximum number of FPGA I/O pairs in mode LVDS is 576. Therefore, the maximum width of connectors is 576.

$$\begin{aligned} &width\,of\,connectors\ *\ number\,of\,connectors \\ &\leq max.\,FPGA\,I/O\,pairs\,in\,LVDS \end{aligned} \tag{5.1}$$

The minimum width of connectors is limited by the characteristics of one FPGA I/O bank.

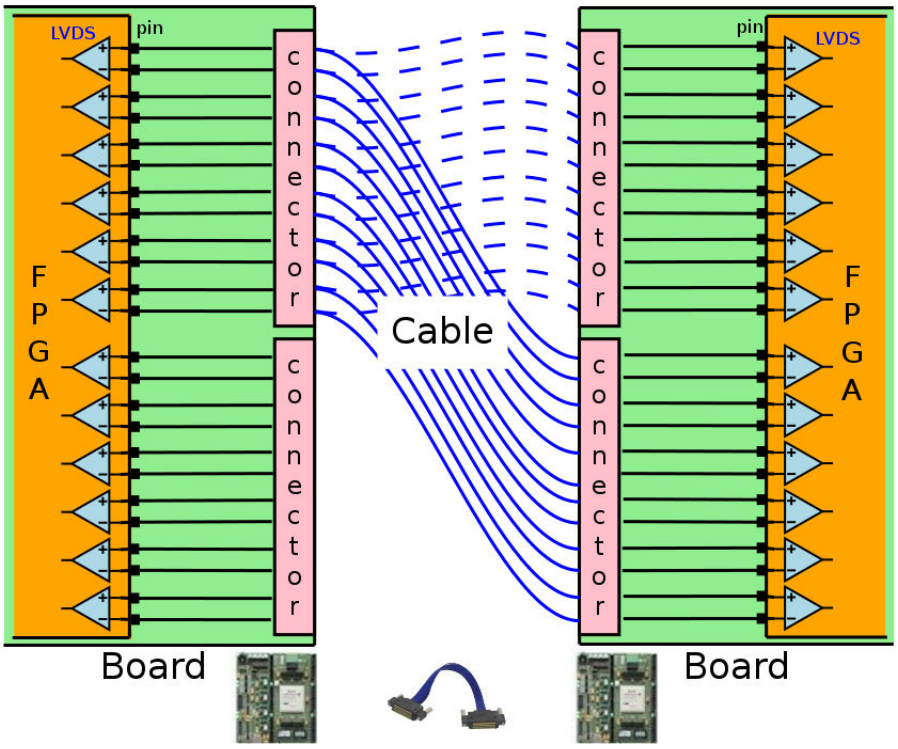


Figure 5.1: *The width of connectors in the cabling platform*

According to [SelectIO, 2014], a bank is a group of FPGA I/O pins that share a common resource such as one power supply or one output current reference. The number of FPGA I/O pairs (in LVDS mode) in a bank is defined as the width of the bank. In the inter-FPGA communication architecture of the cabling platform, one pair of FPGA I/Os per connector is used to propagate the clock when the width of connectors is smaller than the width of the bank. If not, one pair of FPGA I/Os per bank per connector is used to propagate the clock. Due to the limitation of clock capable FPGA I/O pairs in a bank, the width of connectors is limited as shown in Equation 5.2 in order to fully employ all the FPGA I/O pairs (one connector contains at least one pair of clock capable FPGA I/Os). For example, in the Virtex-7 [Package, 2014], a bank has 24 pairs of FPGA I/O pins which can be used as LVDS. Among them, only 4 pairs are clock capable. This results that the minimum width of connectors is 6 inter-FPGA tracks ($\frac{1}{4}$ bank). If the width of connectors is less than this minimum value (i.e. the width of connectors is 5), only 20 pairs of FPGA I/Os will be used in a bank due to that one bank only has 4 clock capable I/O pairs. Therefore, 4 FPGA I/O pairs will be wasted.

$$width\ of\ connectors \geq \frac{the\ width\ of\ the\ bank}{\#\ of\ clock\ capable\ I/O\ pairs} \tag{5.2}$$

The cabling platform benefits from the availability and the customization. The performance of the cabling platform depends on the distribution of the cables and the placement of the external

interfaces. Nevertheless, there is no tool to automatically have a solution for the cable distribution. Today, the cables (resp. the external interfaces) are distributed (resp. placed) according to the experience of board designers. Compared to the off-the-shelf platform, the added value, in terms of performance, of cabling platforms can be heavily impaired by an inefficient cable distribution.

The two existing cabling platforms (proFPGA and HAPS) have different width granularity of the connectors (resp. cables). The connections inter FPGAs as well as connections to external interfaces of the cabling platform are distributed with the granularity of one connector (resp. one cable). The optimal width (granularity) of the connectors need to be explored to achieve the maximum performance in the cabling platform.

5.3 Automatic Design Flow for Creating a Cabling Platform

In the manuscript, we propose an automatic design flow for creating a cabling platform as shown in Figure 5.2, more specifically an algorithm optimizing the distribution of the cables. The proposed automatic design flow that is realized in python programming language, automatically joins the point tools. It consists of the platform generation flow and the implementation flow. In the platform generation flow, a cabling platform tailored for a given design is automatically generated by only choosing the FPGA type used. In the implementation flow, the partitioned design is routed into the generated multi-FPGA platform. As most of off-the-shelf platforms, there is only one FPGA type used in one cabling platform. Therefore, different cabling platforms can be identified by the input design and the FPGA type.

The automatic design flow starts by the platform generation flow. The inputs of the platform generation flow are the prototyped design RTL, the FPGA Lib and the user definition file. Through several steps, the flow generates the cable distribution and the board netlist. The board netlist is high-performance with considering the constraints for high-speed signaling (i.e. LVDS), high performance multiplexing (i.e. ISERDES/OSERDES) and so forth. The FPGA lib contains the logic capacity information and the user I/O information of different FPGA types. The user definition file contains the information of the chosen FPGA type, the maximum FPGA logic capacity utilization, the inter-FPGA track type (In the cabling platform, only 2 point tracks exist), and the multiplexing architecture (In the cabling platform, only ISERDES/OSERDES is used). Different steps of the platform generation flow are presented as follows.

1. Logic Synthesis: The input design is synthesized targeting FPGAs (from RTL to design netlist). This step can be realized by Xilinx XST Synthesis tool [XST, 2014], Altera Quartus Synthesis tool [Quartus, 2014], or third party tools such as Synopsys Synplify tool [Synplify, 2014] and Mentor Graphics Precision tool [Precision, 2014].

2. Estimate the number of FPGAs: The minimum number of FPGAs required in the platform need to be estimated. This step can be realized by the commercial tool waCore provided by the company Flexras Technologies [Flexras, 2014]. As a result, an XML file that contains the information of the number of FPGAs required in the platform is generated.

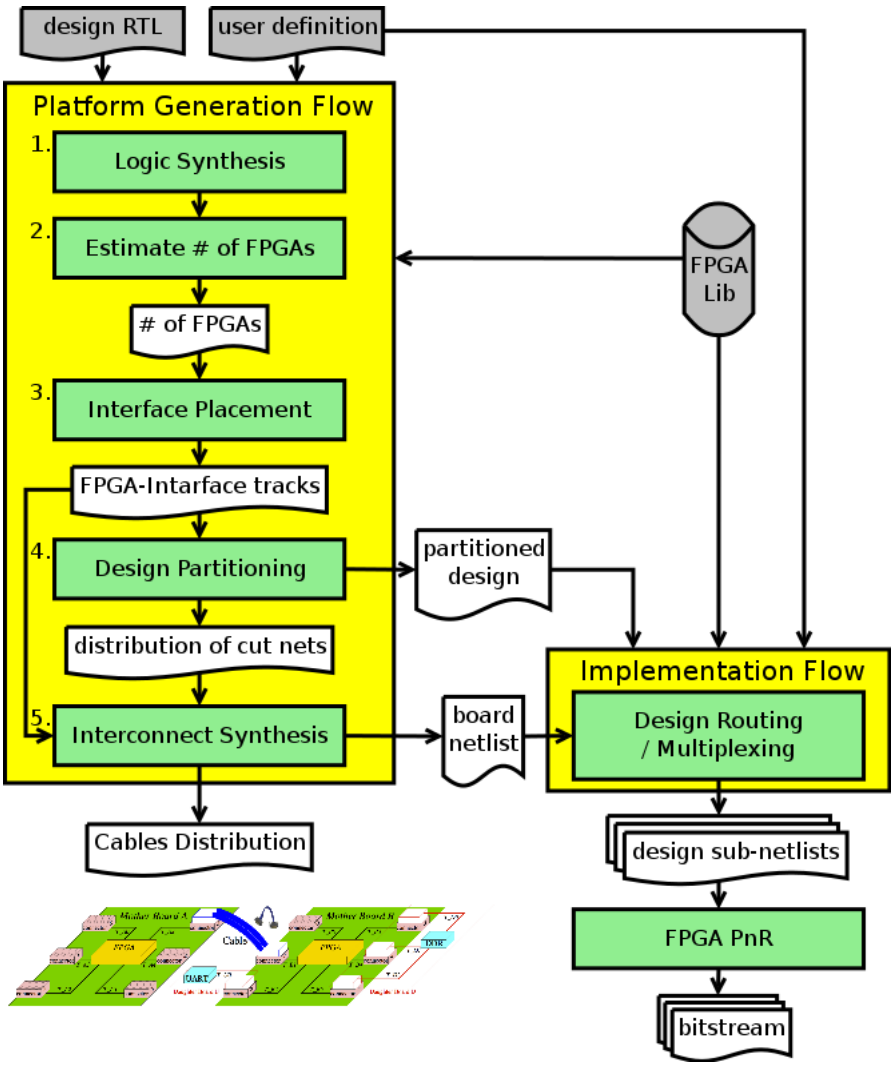


Figure 5.2: *The automatic design flow for creating a cabling platform*

3. Interface Placement: External interfaces to FPGAs are connected. We propose an External Interface Placement Algorithm in Section 4.6 to do this step. The proposed external interface placement algorithm is realized in python programming language. At the end of this step, the FPGA-to-Interface tracks file in XML format is generated.

4. Design Partitioning: This step is done by the commercial tool waPart provided by the company Flexras Technologies [Flexras, 2014]. The partitioning needs to take the connection constraints (the distribution of inter-FPGA connections) into consideration, in order to achieve a higher performance solution. We propose an algorithm to generate a temporary balanced platform in Section 4.7 (realized in python programming language). At the end of this step, the partitioned design netlist in Verilog format and the cut nets distribution file in XML format are generated.

5. Interconnect Synthesis: Inter-FPGA tracks are distributed according to the distribution of cut

nets with the granularity of one connector (resp. one cable). We enhance the Interconnect Synthesis Algorithm in Section 4.8 to do this step. The enhanced part in the Interconnect Synthesis Algorithm is called the cable distribution algorithm, which will be detailed in the following. The enhanced interconnect synthesis algorithm is realized in python programming language. Finally, the board netlist in Verilog format and the cable distribution file in text format are generated.

After finishing the platform generation flow, the automatic design flow passes to the implementation flow. The implementation flow takes the partitioned design netlist, the board netlist, the FPGA lib and the user definition file as inputs. Due to that the design has already been synthesized and partitioned in the platform generation flow, the implementation is to route and multiplex the partitioned design into the generated platform to verify the functionality and to evaluate the performance. We propose a routing algorithm (that enhances the Turki's algorithm [Turki et al., 2013]) in Chapter 3 to do the design routing. The proposed routing algorithm is realized in C++ programming language. It generates a TCL format file that contains the information of the cut net to inter-FPGA track mapping. Then, the commercial tool waRoute provided by the company Flexras Technologies [Flexras, 2014] will read this TCL format file, in order to multiplex several cut nets onto a single track and add the multiplexing IPs in each part of the partitioned design. Finally, the design sub-netlists in Verilog format are generated, which can be loaded by FPGA PnR tools to generate the bitstreams.

The proposed automatic design flow for creating a cabling platform in this Section is similar to the automatic design flow for creating a custom platform proposed in Section 4.9. The difference between these two automatic design flows is the interconnect synthesis algorithm. First, the granularity of distributing inter-FPGA connections is different. The granularity is one connector (resp. one cable) in the cabling platform instead of one inter-FPGA track in the custom platform. Therefore, the available I/Os per FPGA is the number of connectors for the cabling system. Then, the generated results are different. In the custom platform, the interconnect synthesis generates the board netlist in Verilog format and the fsp format file for generating PCB layout. Nevertheless, in the cabling platform, the board netlist in Verilog format and the cable distribution file in text format (instead of a fsp format) are generated. The cable distribution file is a report, which tells the users how to connect the cables (resp. daughter boards) to the corresponding connectors.

5.3.1 Cable Distribution Algorithm

We have proposed the interconnect synthesis algorithm in Section 4.8 to automatically distribute inter-FPGA tracks according to the distribution of cut nets in order to tailor the custom platform for the given design. In this Section, we enhance this algorithm to automatically optimize the distribution of the cables in the cabling platform as shown in Figure 5.3. The enhanced part is called the cable distribution algorithm, located in the left of the figure where "Cabling = Yes". Only 2-point tracks are supported in the cabling platform due to that only 2-point cables exist.

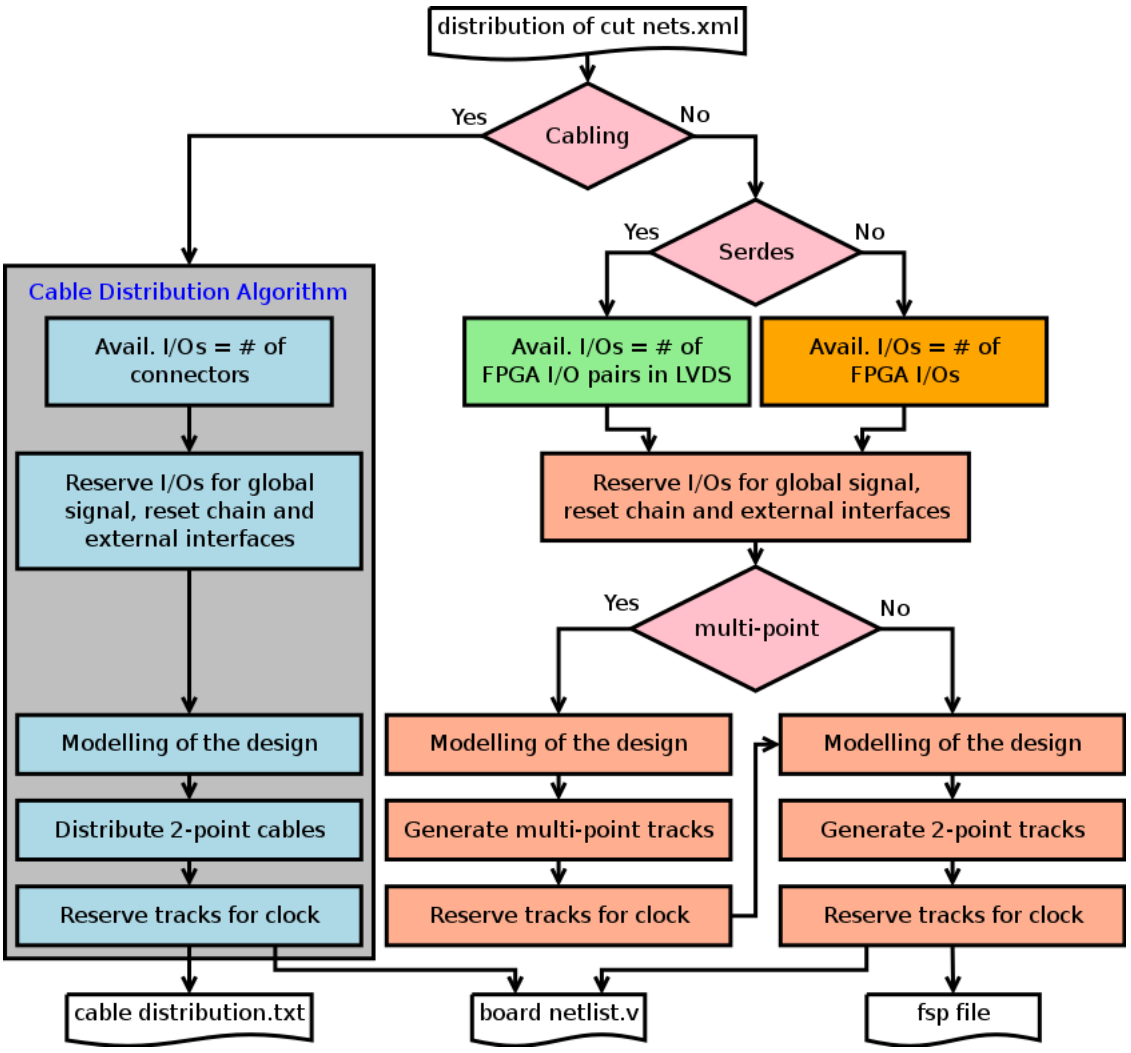


Figure 5.3: *The enhanced interconnect synthesis algorithm*

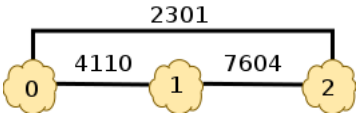
An example, which uses three FPGAs to create a cabling platform for an industrial design, is studied in this Section. The proposed cable distribution algorithm is independent of FPGA types. The chosen FPGA in this example is XC7V2000TFLG1925. In the example, the maximum I/Os per FPGA are 24 (connectors) and each connector has 24 inter-FPGA tracks in mode LVDS (width of connectors).

There are two steps in the cable distribution algorithm. The first step is to reserve one connector for the Global Signals (i.e. global clock, global reset and etc., detailed in Section 4.7.1) and the reset chain in each FPGA, and reserve connectors for the external interfaces in the corresponding FPGAs. In the example, 1 connector is reserved for the Global Signals and the reset chain. Therefore, available I/Os per FPGA are 23. The prototyped design in the example, has 1 DDR as the external interface. According to External Interface Placement Algorithm proposed in Section 4.6, the DDR will be connected to the FPGA 0. Then, after the Design Partitioning, the DDR is connected with the logic in Part 0 (contained in FPGA 0). Therefore, 3 connectors in FPGA

0 need to be reserved for DDR. After that, available I/Os per FPGA are shown in Figure 5.4(a). According to Section 4.8, the design is modelled as Figure 5.4(b). In the design modelling, all the multi-terminal nets will be split in 2-terminal nets. Then, the direction of the cut net is neglected in the modelling.

FPGA	Cable I/Os
FPGA 0	20
FPGA 1	23
FPGA 2	23

(a) available I/Os



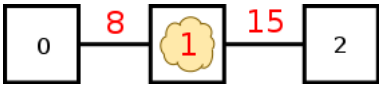
(b) design model

Figure 5.4: *After assigning the global signals, the reset chain and the XIs*

After assigning the global signals, the reset chain and the external interfaces, the next step is to choose one part and then define the inter-FPGA tracks for its corresponding FPGA according to the distribution of cut nets. The critical path of multi-FPGA platforms is the inter-FPGA tracks where the cut nets pass through. Chapter 3 shows that the system clock frequency is limited by the maximum number of cut nets passing through one inter-FPGA track, thus limited by the highest ratio of the number of cut nets to the number of available I/Os. Part 1 is the chosen part due to that Part 1 has the highest ratio (as shown in Figure 5.5(a)). Therefore, inter-FPGA tracks of FPGA 1 will be distributed according to Equation 4.3 of Section 4.8. Part 1 has 11714 total cut nets. Among the total cut nets, there are 4110 cut nets between Part 0 and Part 1 and 7604 cut nets between Part 1 and Part 2. Therefore, $23 * \frac{4110}{11714} = 8$ cables will be distributed between FPGA 0 and FPGA 1, and $23 * \frac{7604}{11714} = 15$ cables will be distributed between FPGA 1 and FPGA 2 as shown in Figure 5.5(b).

Part	cut nets	FPGA	Cable I/Os	Cable ratio
Part 0	6411	FPGA 0	20	321
Part 1	11714	FPGA 1	23	510
Part 2	9905	FPGA 2	23	431

(a) choose the part



(b) the cabling platform

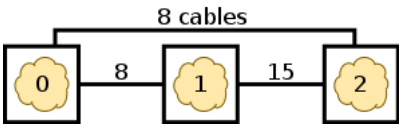
Figure 5.5: *The generation of the platform: Step 1*

After having defined the inter-FPGA tracks of FPGA 1, cut nets of Part 1 will be removed from the design model. Then, the number of cut nets on each part and available I/Os for corresponding FPGAs are updated as shown in Figure 5.6(a). In the next iteration, Part 0 and Part 2 have the same number of cut nets as there are only two parts in the design modelling. Part 2 has less available I/Os, thus higher ratio of the number of cut nets to the number of available I/Os than Part 0. Therefore, inter-FPGA tracks of FPGA 2 will be defined. Part 2 has 2301 cut nets and all the cut nets on Part 2 are cut nets between Part 0 and Part 2. Therefore, 9 cables will be distributed between FPGA 0 and FPGA 2 as shown in Figure 5.6(b).

After having defined the inter-FPGA tracks of FPGA 2, cut nets of Part 2 will be removed from the design model. After that, there is no cut net in the design model and all the inter-FPGA tracks have been defined. The generated cabling platform is shown in Figure 5.6(b), One cable (resp. one connector) in the example means 24 inter-FPGA tracks in mode LVDS. Therefore, in the example,

Part	cut nets	FPGA	Cable I/Os	Cable ratio
Part 0	2301	FPGA 0	12	192
Part 1	0	FPGA 1	0	---
Part 2	2301	FPGA 2	8	288

(a) update cut nets and available I/Os



(b) the cabling platform

Figure 5.6: *The generation of the platform: Step 2*

the width of connectors equals the width of the bank (The chosen FPGA is XC7V2000TFLG1925). As discussed in this Section, one pair of FPGA I/Os per cable is used to propagate the clock when the width of connectors is smaller than the width of the bank. If not, one pair of FPGA I/Os per bank per cable is used to propagate the clock. In order that the inter-FPGA communication is full-duplex (one cut net can be propagated from FPGA A to FPGA B while another cut net from FPGA B to FPGA A), the number of FPGA I/Os used for propagating the clock in one cable need to be multiplied by 2.

5.3.2 Performance Evaluation

The inter-FPGA delay is $1ns$ in the hardwired platform (that includes both the off-the-shelf platform and the custom platform where inter-FPGA tracks are realized by PCB traces). The performance (in terms of the system clock frequency) can be defined as Equation 2.3 of Section 2.3 when implementing ISERDES/OSERDES as the inter-FPGA communication architecture in the hardwired platform. In Equation 2.3, the system clock frequency is the inter-FPGA fast clock frequency divided by the latency. Nevertheless, according to [Posner, 2013a], the inter-FPGA delay T_{inter} is $4ns$ (resp. $5ns$) when the maximum cables are 25cm (resp. 50cm) long in the cabling platform. The inter-FPGA delay is the sum of the FPGA-to-connector delay, the cable delay and the connector-to-FPGA delay as FPGAs are interconnected by cables. The increase in the inter-FPGA delay adds the latency. As the inter-FPGA data rate is $1Gbps$ for all the platforms in case of ISERDES/OSERDES, 3 (resp. 4) inter-FPGA fast clock cycles are added in the latency due to that the inter-FPGA delay increases from $1ns$ to $4ns$ in 25cm long cables (resp. $5ns$ in 50cm long cables). Therefore, the achieved performance calculated in Equation 2.3 will be modified as in Equation 5.3 (resp. Equation 5.4) if the maximum cables are 25cm (resp. 50cm) long in the cabling platform. The maximum number of cut nets passing through one ISERDES/OSERDES is noted as mux_serdes.

$$sys\_clk = \frac{if\_clk}{lat} = \frac{250\ MHz}{10 + ceil(\frac{mux\_serdes}{4})} \tag{5.3}$$

$$sys\_clk = \frac{if\_clk}{lat} = \frac{250\ MHz}{11 + ceil(\frac{mux\_serdes}{4})} \tag{5.4}$$

According to Chapter 3, intermediate FPGAs may be necessary if there is no available track between the FPGA (the Driver) and the FPGA (the Receiver) in Design Routing process. The

maximum number of intermediate FPGAs needed (routing hop) is noted as hop. When implementing ISERDES/OSERDES, the routing hops are not realized in a pipelined way according to Section 3.6.3. The transmission of data in the intermediate FPGA can only start when all the data is received from the Driver FPGA. Therefore, when considering routing hops, the achieved performance calculated in Equation 5.3 (resp. Equation 5.4) will be modified as in Equation 5.5 (resp. as shown in Equation 5.6) if the maximum cables are 25cm (resp. 50cm) long in the cabling platform.

$$sys\_clk = \frac{if\_clk}{lat} = \frac{250\ MHz}{(10 + ceil(\frac{mux\_serdes}{4})) * (1 + hop)} \tag{5.5}$$

$$sys\_clk = \frac{if\_clk}{lat} = \frac{250\ MHz}{(11 + ceil(\frac{mux\_serdes}{4})) * (1 + hop)} \tag{5.6}$$

5.3.3 Time of Availability Evaluation

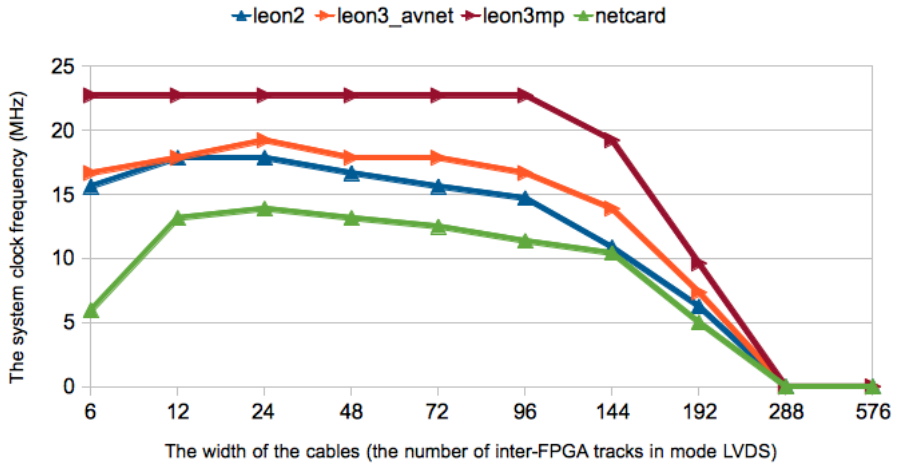
The time of availability is an essential element for FPGA-based prototyping. The earlier the prototyping platform is available, the earlier the developers can start the hardware/software integration. Therefore, the time spent for creating a cabling platform need to be taken into consideration. According to [Amos et al., 2011], when creating the custom platform for a specific design, there are different tasks from design RTL to final PCB board. These tasks include: FPGA circuit development, PCB layout, PCB manufacture, Assembly, Test and Rework. Nevertheless, the cabling platform does not need PCB layout, PCB manufacture, Assembly, Test and Rework. This is because FPGA boards in the cabling platform are ready-made. The cable distribution (inter-FPGA distribution) in the cabling platform is user-defined and need to be tailored for the specific design. This process is the FPGA circuit development (from design RTL to board netlist and cable distribution). According to [Amos et al., 2011], a manual FPGA circuit development (a manual cable distribution) needs 1-2 months. The proposed design flow in this Section for creating a cabling platform automatizes the FPGA circuit development. Therefore, with the help of the proposed automatic design flow, the time of availability for the cabling platform reduces from 1-2 months to several hours.

5.4 Optimal Width of Connectors

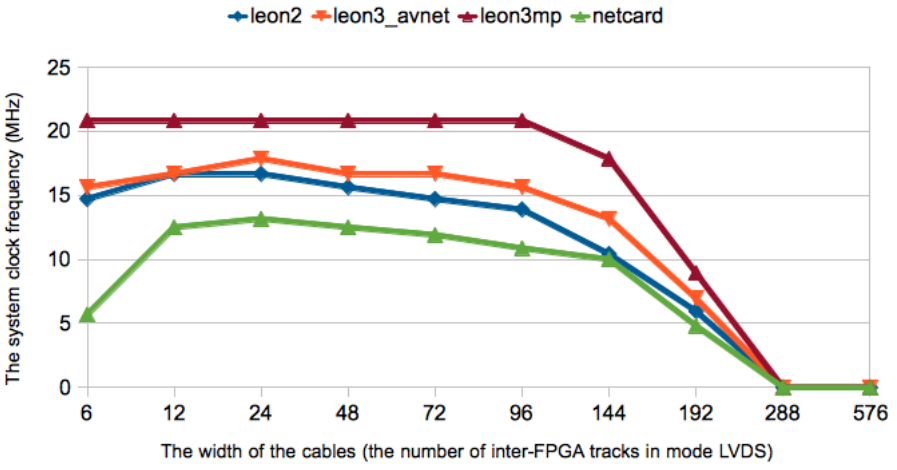
In the experiments, four testbenches (leon2, leon3_avnet, leon3mp, netcard) from Gaisler Research Benchmarks [Gaisler, 2014] will be used. According to our knowledge, these testbenches are the largest available benches representative of industrial designs. The proposed automatic design flow for creating a cabling platform is independent of FPGA types. The chosen FPGA is XC7V2000TFLG1925 due to that it is the latest FPGA. In order to be comparable with the latest off-the-shelf platform presented in Figure 3.1 that use the same FPGA type, the cabling platform has 4 FPGAs. As the XC7V2000TFLG1925 is the largest FPGA, the logic capacity utilization rate

had been reduced between 10% and 20% to force each testbench to fill in 4 FPGAs, while keeping the design internal structure (connections) unaltered.

Experiments explore the optimal width of connectors in the cabling platform to achieve higher performance. According to Section 5.2, the minimum width of connectors is 6 ($\frac{1}{4}$ bank) and the maximum width is 576. The number of the cables should be an integer, and the Equation 5.1 and the Equation 5.2 should be fulfilled. Each testbench is implemented in different cabling platforms with different width of connectors as shown in Figure 5.7. These results are obtained by changing the parameter controlling the width of connectors in the automatic design flow and relaunching the flow to get the achieved performances. The runtime to run the whole flow is about several hours.



(a) length of the cables = 25cm



(b) length of the cables = 50cm

Figure 5.7: *The influence of the width of connectors in performance*

For all the testbenches, the results show that the optimal system clock frequency can be achieved when the width of connectors is 24 inter-FPGA tracks (1 bank) in the cabling platform. When the width of connectors is less than 1 bank, more FPGA I/Os will be used for propagating the clock instead of the data, thus the system clock frequency may degrade. When the width of connectors is more than 1 bank, the interconnect synthesis has less adjustability to distribute inter-

FPGA tracks due to the coarser granularity. In this case, the generated platform can be not adapted for the given design and the system clock frequency can degrade. As shown in Figure 5.7, when the width of connectors exceeds 96 (4 banks), the system clock frequency decreases rapidly. The system clock frequency attains 0 when the width of connectors is more than 288 (12 banks, only 2 cables per FPGA in this case) because the design is not routable in the generated platform. In mechanics, cables and connectors of 1-bank width are feasible [HAPS, 2014]. In the following, the cabling platform is established with the optimal width of connectors (1 bank).

5.5 Proposed Cabling Platform

In this manuscript, we propose a cabling platform where one board is composed of one FPGA and several connectors. All the connectors have the same width and the width of connectors is 1 bank (i.e. 24 inter-FPGA tracks in mode LVDS for Virtex-7 FPGA). All the FPGA I/Os are used for FPGA-to-connector connections. The connections between FPGAs as well as the connections to external interfaces can be added or removed by only connecting or disconnecting the cables (resp. daughter boards) with or from the connectors in order to be tailored for the given design.

An example of the proposed cabling platform is shown in Figure 5.8. One board can be connected with other FPGA boards by cables and connectors, and external interface daughter boards by connectors (connectors are capable to support the speed of external interfaces, i.e. DDR [Posner, 2013b]). If the prototyped design has external interfaces (i.e. DDR), the external interfaces will be connected with one FPGA by connectors. If there is no external interface, these connectors will serve as inter-FPGA tracks by connecting them with cables. Therefore, in this case, there is no FPGA I/O wasted in the cabling platform compared to the off-the-shelf platform. (i.e. T_B3, T_B4, T_B5 in the mother board B that are connected with the daughter board D can be served as inter-FPGA tracks by connecting the connector 0, 1, 2 in the Mother Board A with the connector 3, 4, 5 in the Mother Board B). Different from the off-the-shelf platform where specific external interfaces have been fixed to FPGAs, external interfaces in the cabling platform can be connected to any FPGAs by connectors. Therefore, the optimum placement of external interfaces can be explored in order to achieve higher performance (in terms of the system clock frequency) according to Section 4.6 and the partitioning process can be less constrained compared to the off-the-shelf platform. Nevertheless, when the external interface comes to be the external interfaces that contains few pins (i.e. UART only has 2 pins, meaning that 1 track in mode LVDS), one connector is occupied and some FPGA I/Os in the connector are wasted. (i.e. T_B2 has 24 tracks in mode LVDS, but T_U0 has only 1 track and 23 tracks in T_B2 are wasted)

The proposed cabling platform has a flexibility. Moving the cabling platform customized for a specific design to a cabling platform customized for another specific design can be easily done by using the same platform with only re-launching the automatic design flow and re-connecting the cables, if these two specific designs have the same logic capacity size. If the logic capacity sizes of these two specific designs are different, ready-made boards can be easily added to or removed

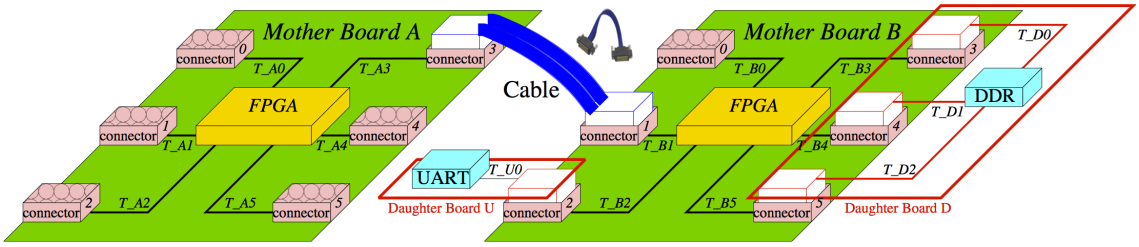


Figure 5.8: *The cabling multi-FPGA platform*

from the cabling platform. Then, the automatic design flow is re-launched and the cables are re-connected.

5.6 Conclusion

In this Chapter, the cabling platform is discussed. The cabling platform benefits from the availability and the customization. Nevertheless, there is no tool to automatically have a solution for the cable distribution. Today, the cables (resp. the external interfaces) are distributed (resp. placed) according to the experience of board designers. Compared to the off-the-shelf platform, the added value, in terms of performance, of cabling platforms can be heavily impaired by an inefficient cable distribution. The two existing cabling platforms (proFPGA and HAPS) have different width granularity of the connectors (resp. cables). The connections inter FPGAs as well as connections to external interfaces of the cabling platform are distributed with the granularity of one connector (resp. one cable). The optimal width (granularity) of the connectors need to be explored to achieve the maximum performance in the cabling platform.

The main contribution is threefold. First, we propose an automatic design flow for creating a cabling platform, more specifically an algorithm optimizing the distribution of the cables. Then, with the help of the proposed automatic design flow, the optimal width of connectors for a cabling platform is explored. Experiments are conducted using four testbenches from Gaisler Research Benchmarks. The results show that the optimal width of connectors is 1 bank in the cabling platform. Finally, we propose a cabling platform where one board is composed of one FPGA and several connectors. All the connectors have the same width and the width of connectors is 1 bank. All the FPGA I/Os are used for FPGA-to-connector connections. The connections inter FPGAs as well as the connections to external interfaces can be added or removed by only connecting or disconnecting the cables (resp. daughter boards) with or from the connectors.

The next Chapter compares three different multi-FPGA platforms (off-the-shelf, custom, cabling). With the help of the proposed automatic tools, the performance gains between these platforms are quantified.

Chapter 6

Comparison of the Different Platforms

6.1 Introduction

The previous three Chapters have discussed the three different multi-FPGA platforms (off-the-shelf, custom, and cabling), and have presented and proposed automatic flows for different platforms. In Section 2.2.4, these three multi-FPGA platforms have been compared qualitatively in terms of availability, performance, flexibility, and cost. With the developed automatic tools, this Chapter will re-compare these three platforms, and therefore quantify the performance gain between those platforms.

6.2 Performance Comparison of Different Multi-FPGA Platforms

The workflow of the performance comparison when implementing a specific design respectively into three different multi-FPGA prototyping platforms, is shown in Figure 6.1 and presented as follows.

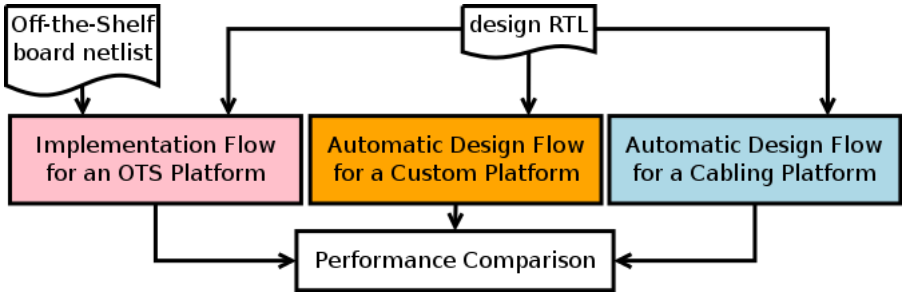


Figure 6.1: *The workflow of the performance comparison*

- When the design is implemented in the off-the-shelf platform, the automatic implementation flow, detailed in Figure 3.2 of Chapter 3, is used.

- When the design is implemented in the custom platform, the automatic design flow for creating a custom platform, detailed in Figure 4.25 of Chapter 4, is used.
- When the design is implemented in the cabling platform, the automatic design flow for creating a cabling platform, detailed in Figure 5.2 of Chapter 5, is used.

Experiments are conducted using four testbenches (leon2, leon3_avnet, leon3mp, netcard) from Gaisler Research Benchmarks [Gaisler, 2014]. The automatic flows are independent of FPGA types. For the comparison, all the multi-FPGA platforms use the same type of FPGAs (XC7V2000TFLG1925) and have 4 FPGAs. Each testbench is implemented respectively in three different multi-FPGA prototyping platforms to compare the performance as shown in Table 6.1. The cabling platform is established with the optimal width of the cables (1 bank). There are two different types of cabling platforms depending on the length of the cables (25cm and 50cm). The performance (in terms of the system clock frequency) is noted as sys_clk. The unit for the system clock frequency is MHz.

Table 6.1: Performance Comparison of Different Platforms

testbench	Off-the-Shelf	Cabling				Custom			
		cables length = 50cm		cables length = 25cm					
	sys_clk (MHz)	sys_clk (MHz)	gain vs Shelf	sys_clk (MHz)	gain vs Shelf	sys_clk (MHz)	gain vs Shelf	gain vs Cabling (50cm)	gain vs Cabling (25cm)
leon2	11.36	16.67	47%	17.86	57%	20.83	83%	25%	17%
leon3_avnet	15.63	17.86	14%	19.23	23%	22.73	45%	27%	18%
leon3mp	25	20.83	-16%	22.73	-9%	27.78	11%	33%	22%
netcard	6.58	13.16	100%	13.89	111%	14.71	124%	12%	6%

The results show that the system clock frequency of the multi-FPGA platforms is about several to tens of MHz for all the testbenches as shown in Figure 6.2. The custom platform can achieve the highest performance compared to the other platforms. The performance gain of the custom platform is up to 124% compared to the off-the-shelf platform. This is due to that inter-FPGA tracks of the off-the-shelf are fixed and generic, and inter-FPGA tracks of the custom platform are distributed to be tailored for the given design. The performance gain of the custom platform is up to 22% (resp. 33%) over the cabling one in 25cm cable length (resp. 50cm cable length). This is because the inter-FPGA delay of the custom platforms ($1ns$) is less than the cabling platform delay ($4ns$ for 25cm cable length and $5ns$ for 50cm cable length). In most of testbenches (leon2, leon3_avnet, netcard), the cabling platform can achieve higher performance compared to the off-the-shelf platform. The performance gain of the cabling platform is up to 111% in 25cm cable length (resp. 100% in 50cm cable length). This is due to that inter-FPGA tracks of the cabling platform are also distributed to be tailored for the given design. Nevertheless, in the testbench (leon3mp), the cabling platform achieves lower performance compared to the off-the-shelf platform. The performance gain of the cabling platform is -9% in 25cm cable length (resp. -16% in

50cm cable length). This is because the cable distribution algorithm is not able to find a better distribution than the off-the-shelf (the worst case being the same distribution as the off-the-shelf platform due to the same constraints for both platforms explained in Section 2.3), and the inter-FPGA delay of the cabling platform is higher than the off-the-shelf platform inter-FPGA delay ($1ns$).

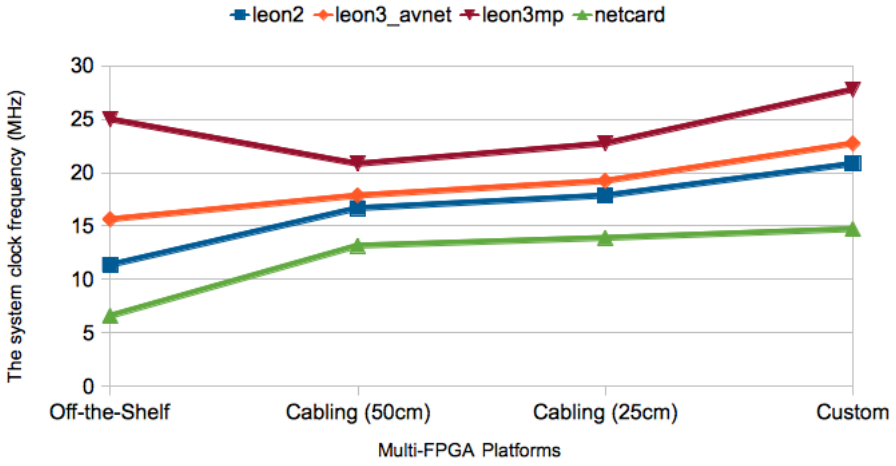


Figure 6.2: *Comparison of the performance*

6.3 Quantitative and Qualitative Comparison of Different Platforms

In Section 2.2.4, these three multi-FPGA platforms are compared qualitatively in terms of availability, performance, flexibility, and cost. With the developed automatic tools, the qualitative comparison is updated in Table 6.2.

Table 6.2: Updated Qualitative Comparison of Different Multi-FPGA Prototyping Platforms

Multi-FPGA Prototyping Platforms	Availability	Performance			Flexibility	Cost	
		Inter-FPGA data rate	Inter-FPGA delay	tracks distribution		Unit Price	Deployment Cost
Hardwired Off-the-Shelf	instantaneous	~ 1Gbps	1ns	generic & balanced	medium	low	high
Cabling	2-12 hours	~ 1Gbps	4ns or 5ns	tailored for the design	high	low	high
Hardwired Custom	3-5 months	~ 1Gbps	1ns	tailored for the design	low	high	low

With the proposed automatic tools, the time of availability for the cabling platform reduces from 1-2 months to several hours. Nevertheless, for the custom platform, the proposed design flow only automatizes the FPGA circuit development. Therefore, 3-5 months are still needed due to the PCB layout and the PCB fabrication. The achieved performance of multi-FPGA platforms depends on the inter-FPGA data rate, inter-FPGA tracks distribution, and inter-FPGA delay. The

automatic design flow for creating a custom platform can generate high-performance board netlist with considering the constraints for high-speed signaling (i.e. LVDS), high performance multiplexing (i.e. ISERDES/OSERDES) and so forth. Therefore, the inter-FPGA data rate of the custom platform that before depended on the designers' experience, today is the same with the off-the-shelf and the cabling platform that are ready-made by the FPGA experts in the commercial companies. The proposed automatic design flows for creating a cabling and custom platform optimize the distribution of inter-FPGA tracks. Therefore, the inter-FPGA tracks distributions of the cabling and custom platform that before depended on the designers' experience, today are tailored for the given design. In the following, all these three multi-FPGA platforms are compared mutually in the quantified performance shown in Table 6.1 with considering the advantages and disadvantages of each platform shown in Table 6.2.

6.3.1 Cabling vs Off-the-Shelf

The inter-FPGA delay of the off-the-shelf platform ($1ns$) is much lower than the cabling platform ($4ns$ or $5ns$). This means that, if the cable distribution algorithm is not able to find a better distribution than the off-the-shelf (the worst case being the same distribution as the off-the-shelf platform due to the same constraints for both platforms explained in Section 2.3), only in this case, the off-the-shelf would give a better performance (i.e. leon3mp). In other cases, the cabling platform always outperforms the equivalent generic off-the-shelf platform in performance (25cm cable length: from 23% gain up to 111%; 50cm cable length: from 14% gain up to 100%). With the automatic tool, the time of availability for the cabling platform is only 2-12 hours. And for the other characteristics such as flexibility and cost, the cabling platform can outperform or equal the off-the-shelf platform as shown in Table 6.2. Therefore, the cabling platform is now an attractive alternative compared to the off-the-shelf platform.

6.3.2 Custom vs Off-the-Shelf

The custom platform can achieve higher performance than the off-the-shelf platform (up to 124% gain). Nevertheless, the proposed automatic design flow for creating the custom platform, is only automatizing the FPGA circuit development and generating the board netlist. According to Table 6.2, making a custom multi-FPGA PCB board needs about 3-5 months (time of availability). Therefore, if only few pieces are needed, and the performance difference is not relevant enough for the user (e.g. 25MHz for leon3mp in the off-the-shelf and 27.78MHz for the custom), selecting the off-the-shelf scenario allows to run tests 3-5 months ahead at 25MHz. Nevertheless, the cost of the custom platform can be lower when a large quantity of platforms is needed (deployment cost). The proposed automatic design flow helps the designers to predict the achieved performance in the custom platform and choose the optimum prototyping platform according to their specifications (board exploration). Therefore, choosing an off-the-shelf or a custom is a tradeoff depending on designers' specifications.

6.3.3 Custom vs Cabling

The performance achieved with the custom platform exceeds the cabling platform. However, the gain is limited (max. 22% gain for 25cm cable length and max. 33% gain for 50cm cable length). Indeed, the availability of the custom platform is about 3-5 months which is much longer than the cabling platform, which is only 2-12 hours. On the flexibility side, the custom platform designed for leon2 may be not adapted for netcard. The flexibility criteria favors the cabling platform. For example, moving the cabling platform customized for a specific design (i.e. leon2) to a cabling platform customized for another specific design (i.e. netcard) can be easily done by using the same platform with only re-launching the automatic design flow and re-connecting the cables. If the logic capacity sizes of these two specific designs are different, ready-made boards can be easily added to or removed from the cabling platform. Then, the automatic design flow is re-launched and the cables are re-connected. Therefore, if the performance or the deployment cost are not stringent constraints, the cabling platform can be a better choice with the proposed automatic design flow.

6.4 Conclusion

In Section 2.2.4, these three multi-FPGA platforms (off-the-shelf, custom, and cabling) have been compared qualitatively in terms of availability, performance, flexibility, and cost. With the developed automatic tools, this Chapter has re-compared these three platforms, and therefore quantified the performance gain between those platforms. Experiments are conducted using four testbenches from Gaisler Research Benchmarks. The results show that choosing an off-the-shelf or a custom is still a tradeoff depending on designers' specifications. Nevertheless, apart from some stringent constraints (such as deployment cost or specific frequency needed), the relatively new cabling paradigm with the automatic, cable distribution tool, offers an attractive alternative compared to the two other platforms.

The next Chapter concludes this manuscript and lists the future lines of research.

Chapter 7

Conclusion and Future Work

7.1 Summary of the Thesis

The utility of PFGA-based prototyping is undisputed. It permits hardware designers to develop and test their systems, and gives software developers early access to a fully functioning hardware platform. As the complexity of System on Chip (SoC) circuits and the quantity of software to be developed are increasing, the software developers can no longer wait for the chip to be fabricated for the hardware/software integration in order to meet the ever-shrinking time-to-market window. Thus, FPGA-based prototyping is no longer optional. As SoC/ASIC has a higher silicon area overhead than FPGA, multi-FPGA prototyping platforms are indispensable.

Even though the logic capacity and the number of FPGA I/Os are increasing generation after generation, the logic capacity grows at a much higher rate than the number of FPGA I/Os. Thus, FPGA I/Os are becoming a scarce resource, worsening the inter-FPGA bandwidth generation after generation. Unfortunately, multi-FPGA platforms suffer from large timing delays in inter-FPGA communication compared to intra-FPGA net delays. Therefore, it becomes more and more difficult to prototype an SoC/ASIC design at a proper performance.

We have classified multi-FPGA prototyping platforms in three categories: off-the-shelf, custom, and cabling. The off-the-shelf platform consists of a ready-made generic multi-FPGA board, where all the inter-FPGA connections are fixed and realized using PCB traces. The custom platform consists of a build-your-own specific multi-FPGA board, where all the inter-FPGA connections are realized using PCB traces as well. The cabling platform consists of multiple ready-made FPGA boards connected by cables and connectors, where all the inter-FPGA connections are realized using cables and connectors instead of PCB traces.

Off-the-shelf platforms come with two types of inter-FPGA tracks: 2-point tracks or multi-point tracks. However, existing routing algorithms can not automatically employ multi-point tracks. In the manuscript, we have proposed a new routing algorithm to spare FPGA I/Os by automatically routing and multiplexing multi-terminal nets in multi-point tracks. The experiment results in an off-the-shelf platform of six Virtex-5 show up to 22% gain in performance, compared

to the existing routing algorithm that routes multi-terminal nets of the design through a sequence of 2-point tracks and considers multi-point tracks as 2-point tracks only.

70% of multi-FPGA prototyping platforms are custom platforms due to performance requirement, external interfaces, and cost. Different from the off-the-shelf platform that has generic and balanced connections, the connections inter FPGAs as well as the connections to external interfaces of the custom platform are user-defined and tailored for a specific design. Naturally, the custom platforms should achieve the highest performance. Nevertheless, crafting a custom multi-FPGA platform is today a manual process. The performance and the cost of the platform depend on the FPGA expertise and SoC DUT knowledge of the prototyping team. Therefore, not so good performance platforms commonly occur. Crafting a custom platform is an iterative process, thus is time-consuming (4-7 months). The process involves FPGA circuit development (from design RTL to board netlist), PCB layout, PCB manufacture, Assembly, Test and Rework. The tradeoff between the performance and the cost exists due to that the performance of multi-FPGA platforms is limited by the inter-FPGA communications. The board exploration with different FPGA types in FPGA circuit development can not be done to design an optimum prototyping platform because one trial needs 1-2 months. As the ratio between the logic capacity and the number of FPGA I/Os is increasing at an exponential rate, it becomes tougher and tougher to keep performance as it was in multi-FPGA platforms. In this manuscript, we have proposed an automatic design flow for creating a custom platform, thus increasing the productivity, enabling board exploration, and optimizing cost and performance. The proposed automatic design flow automatically joins the commercial or self-developed point tools. We have developed several point tools such as: the external interface placement tool that automatically find a solution for the external interface placement, and the interconnect synthesis tool that automatically distribute inter-FPGA tracks according to the distribution of cut nets. The proposed automatic design flow drops the entry barrier of board designers, and can generate high-performance board netlist with taking the high-speed signaling, high-performance multiplexing into consideration. With the proposed automatic design flow, the board netlist of the custom platform can be generated with an acceptable CPU time (2-12 hours) and an acceptable internal memory resource (2-4 GB). This means that the time spent for FPGA circuit development is reduced from 1-2 months to 2-12 hours. Therefore, the board exploration becomes feasible. Designers can choose the most adapted multi-FPGA board according to their specifications. The experiment results also show that the smaller FPGA that is less expensive could be used to reduce the overall cost of the platform with sometimes higher performance.

The cabling platform, which is a comparatively new notion compared to other two platforms, is in between the off-the-shelf and the custom platform. It is semi off-the-shelf due to that it consists of multiple ready-made boards, and semi custom due to that its connections inter FPGAs as well as connections to external interfaces are user-defined and tailored for a specific design. Therefore, the cabling platform benefits from both the availability and the customization. The performance of the cabling platform depends on the distribution of the cables and the placement of the external interfaces. Nevertheless, there is no tool to automatically have a solution for the cable distribution. Today, the cables (resp. the external interfaces) are distributed (resp. placed)

according to the experience of board designers. A high-performance cable distribution becomes more and more arduous because of the limited number of FPGA I/Os. Furthermore, the added value in performance of cabling platforms can be heavily weakened by an inefficient board design. In this manuscript, we have proposed an automatic design flow for creating a cabling platform, more specifically an algorithm optimizing the distribution of the cables. Then, with the help of the proposed automatic design flow, the optimal width of connectors for a cabling platform is explored. The experiment results show that the optimal width of connectors is 1 bank in the cabling platform. Finally, we have proposed a cabling platform where one board is composed of one FPGA and several connectors. All the connectors have the same width and the width of connectors is 1 bank. All the FPGA I/Os are used for FPGA-to-connector connections. The connections inter FPGAs as well as the connections to external interfaces can be added or removed by only connecting or disconnecting the cables (resp. daughter boards) with or from the connectors.

After finishing the discussions above on the three different categories of multi-FPGA platforms (off-the-shelf, custom, and cabling), board designers may have a question. Which category is the optimum solution according to designers' specification? Different categories of platforms had already been compared qualitatively in terms of availability, performance, flexibility, and cost. With the developed automatic tools, these three platforms are re-compared. The experiment results show that choosing an off-the-shelf or a custom is still a tradeoff depending on designers' specifications. The proposed automatic design flow integrates the expertise for designing a high-performance board netlist. Thus, the performance of the custom platform that before depended on the designers' experience, can always be better than the off-the-shelf platform. Nevertheless, the proposed design flow only concerns FPGA circuit development. 3-5 months are still needed for creating a custom platform owing to PCB layout and PCB fabrication. When referring to the cabling platform, the proposed design flow reduces the time of availability from 1-2 months to 2-12 hours because the PCB of cabling platforms is ready-made. Furthermore, the experiment results show that generally the cabling platform achieves higher performance than the off-the-shelf except the only case that the cable distribution algorithm is not able to find a better distribution (the worst case being the same distribution as the off-the-shelf platform). Therefore, the cabling platform is now more attractive than the off-the-shelf platform. Finally, we compare the cabling platform with the custom platform. The experiment results show that the custom platform achieves a limited performance gain with much longer time of availability. On the flexibility side, the custom platform tailored for one design may be not adapted for another design. Thus, another platform has to be created and at least 3-5 months need to be spent. Nevertheless, moving the cabling platform tailored for a specific design to a cabling platform customized for another specific design can be easily done by using the same platform with only re-launching the automatic design flow and re-connecting the cables. If the logic capacity sizes of these two specific designs are different, ready-made boards can be easily added to or removed from the cabling platform. Then, the automatic design flow is re-launched and the cables are re-connected. It takes only 2-12 hours. Therefore, if the performance or the deployment cost are not rigorous constraints, the cabling platform can be a better choice with the proposed automatic design flow.

7.2 Future Work

Before proposing different research axes, it is interesting to have a look at the FPGA's evolution especially the I/Os interface as it is the bottleneck for multi-FPGA prototyping.

7.2.1 Evolution of FPGAs and Analysis

In Figure 7.1, we can classify FPGA I/O pins into two types: user I/Os and MGT I/Os. In this manuscript, Xilinx Virtex Family is studied according to [Virtex-4, 2014] [Virtex-5, 2014] [Virtex-6, 2014] [Virtex-7, 2014]. The results show that the evolution of the data rate capability for user I/Os and MGT I/Os is different.

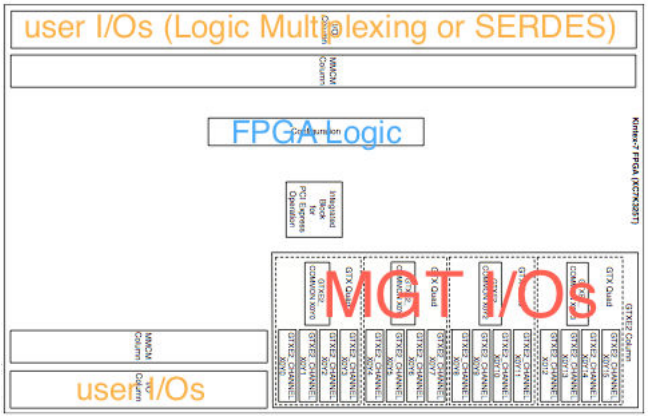


Figure 7.1: *MGT in an FPGA*

The total user I/Os data rate equals the data rate of one user I/O multiplied by the number of user I/Os. In one Virtex Family (for example: Virtex-7), there are different devices (for example: 2000T, X485T and etc.). Different devices in the same Virtex Family can have different total user I/Os data rate values. The maximum total user I/Os data rate in one Virtex Family is the maximum value of them. The maximum total user I/Os data rate increases from Virtex-4 to Virtex-5 as shown in Figure 7.2. Nevertheless, it stays stable from Virtex-5 to Virtex-7.

The total MGT I/Os data rate equals the data rate of one MGT I/O multiplied by the number of MGT I/Os. Different devices in the same Virtex Family can have different total MGT I/Os data rate values. The maximum total MGT I/Os data rate in one Virtex Family is the maximum value of them. The maximum total MGT I/Os data rate increases from Virtex-4 to Virtex-7 as shown in Figure 7.3. Recently, this increase accelerates from Virtex-6 to Virtex-7.

In conclusion, the data rate of user I/Os that is used for multi-FPGA prototyping, is at a stop. Conversely, the data rate of MGT I/Os that is not actually used for multi-FPGA prototyping, is increasing.

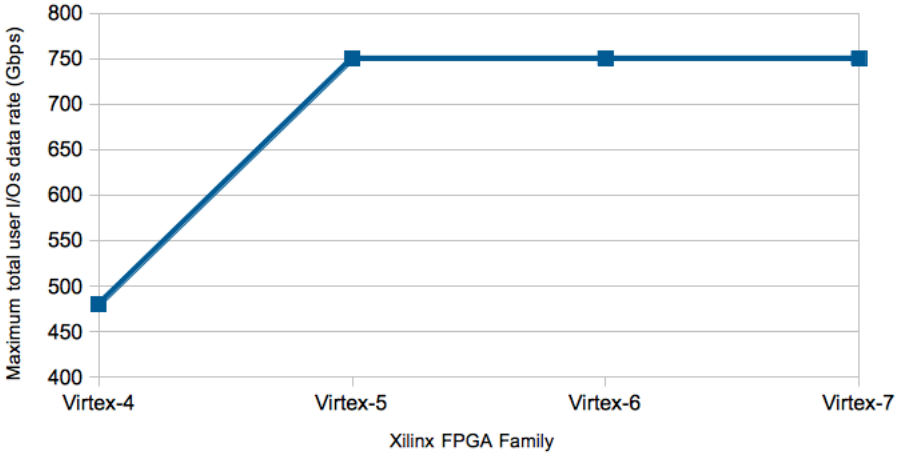


Figure 7.2: *Evolution of the maximum total user I/Os data rate*

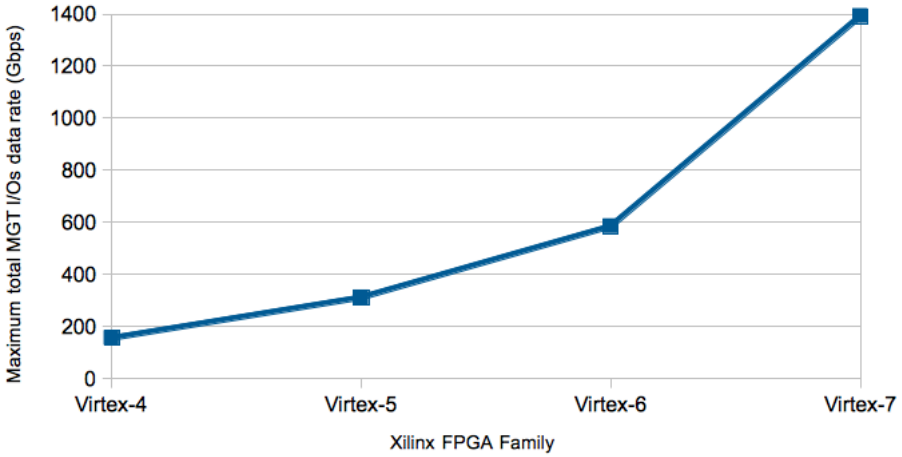


Figure 7.3: *Evolution of the maximum total MGT I/Os data rate*

7.2.2 Future Research

Improving Time-Division-Multiplexing Techniques: Due to that the user I/Os data rate is at a standstill, the future work will target at increasing the user I/Os data rate. As the total MGT I/Os data rate increases rapidly, it is becoming more and more beneficial in employing MGT I/Os if possible. In [Tang et al., 2014], we explore the ability of MGT I/Os in multi-FPGA based prototyping. The results show that the system clock frequency (performance) is limited to ~ $7MHz$ when using MGT I/Os due to the high latency of MGT. Moreover, only a few MGTs are available due to its silicon area is large. Therefore, the future work will also target at reducing the latency and the silicon area of MGT I/Os.

Design-aware Automatic Flow: The proposed automatic design flow for creating a custom and a cabling platform needs to choose an FPGA type as input. The future work will study the ar-

chitecture of the given design (i.e. the design's Rent's Rule and the design's hierarchy) and the architecture of FPGAs (among all the FPGAs in FPGA lib) to predict the most adapted FPGA in terms of performance.

Heterogeneous Multi-FPGA Platform: All the FPGAs in one custom platform (or one cabling platform) generated by the proposed automatic design flow is the same FPGA type. A heterogeneous multi-FPGA platform may be existing to reduce the cost while increasing or keeping the same performance. The future work will examine the generation of a heterogeneous platform by the automatic design flow for creating a custom and a cabling platform.

Automatic PCB Design Flow for a Custom Platform: The proposed automatic design only automates the process of FPGA circuit design (from a given design to the board netlist). Therefore, creating a custom platform still needs several months due to the PCB layout and the PCB fabrication. The future work will automate the PCB design flow (from the high-performance board netlist to the high-performance PCB layout, taking high-speed signaling into consideration).

List of Publications

International Conference Publications

Regular Presentations

1. **Performance Comparison between Multi-FPGA Prototyping Platforms: Hardwired Off-the-Shelf, Cabling and Custom**
   Qingshan Tang, Matthieu Tuna and Habib Mehrez
   in Proceedings of International Symposium on Field-Programmable Custom Computing Machines (**FCCM'14**), Boston, MA, USA, May 11-13, 2014, Page 125-132
   Acceptance rate: 16%

2. **Multi-FPGA Prototyping Board Issue : the FPGA I/O Bottleneck**
   Qingshan Tang, Matthieu Tuna and Habib Mehrez
   in Proceedings of International Conference on Embedded Computer Systems : Architectures, Modeling, and Simulation (**SAMOS'14**), Samos, Greece, July 14-17, 2014, Page 207-214

3. **Routing Algorithm for Multi-FPGA Based Systems Using Multi-Point Physical Tracks**
   Qingshan Tang, Matthieu Tuna and Habib Mehrez
   in Proceedings of the 24th IEEE International Symposium on Rapid System Prototyping (**RSP'13**), Montreal, Canada, October 3-4, 2013, Page 2-8

4. **Design for prototyping of a parameterizable cluster-based Multi-Core System-on-Chip on a Multi-FPGA board**
   Qingshan Tang, Matthieu Tuna and Habib Mehrez
   in Proceedings of the 23rd IEEE International Symposium on Rapid System Prototyping (**RSP'12**), Tampere, Finland, October 11-12, 2012, Page 71-77

Poster Presentations

5. **Future Inter-FPGA Communication Architecture for Multi- FPGA Based Prototyping**
   Qingshan Tang, Matthieu Tuna and Habib Mehrez
   in Proceedings of the ACM/SIGDA 18th International Symposium on Field Programmable Gate Arrays (**FPGA'14**), Monterey, CA, USA, February 26-28, 2014, Page 251

6. **Towards High Performance GHASH for Pipelined AES-GCM Using FPGAs**
   Karim Abdellatif, Roselyne Chotin-Avot, Habib Mehrez and Qingshan Tang
   in Proceedings of the ACM/SIGDA 22nd International Symposium on Field Programmable Gate Arrays (**FPGA'14**), Monterey, CA, USA, February 26-28, 2014, Page 242

7. **Automatic Design Flow for Creating a Custom Multi-FPGA Board Netlist**
   Qingshan Tang, Matthieu Tuna, Zied Marrakchi and Habib Mehrez
   in Proceedings of the 9th International Symposium on Applied Reconfigurable Computing (**ARC'13**), Los Angeles, CA, USA, March 25-27, 2013, Page 221

French Conference Publications

8. **Flot de Conception pour la Génération Automatique de Carte Multi-FPGA**
   Qingshan Tang
   dans les 16èmes Journées Nationales du Réseau Doctoral en Microélectronique (**JNRDM'13**), Grenoble, France, Juin 10-12, 2013

Bibliography

[Alpert et al., 1997] Alpert, C., Huang, J., and Kahng, A. (1997). Multilevel Circuit Partitioning. In *Proceedings of the 34th Conference on Design Automation*, pages 530–533, Anaheim, CA, USA. 47

[Altera, 2014] Altera (2014). FPGA CPLD and ASIC from Altera. Available from: `http://www.altera.com`. 13, 25, 31, 61, 67, 93

[Amos et al., 2011] Amos, D., Lesea, A., and Richter, R. (2011). *FPGA-Based Prototyping Methodology Manual*. Synopsys. 10, 13, 14, 32, 34, 35, 36, 37, 62, 77, 90, 91, 92, 112

[Asaad et al., 2012] Asaad, S., Bellofatto, R., Brezzo, B., Haymes, C., Parker, M., Roewer, T., Saha, P., Takken, T., and Tierno, J. (2012). A Cycle-accurate, Cycle-reproducible multi-FPGA System for Accelerating Multi-core Processor Simulation. In *Proceedings of the ACM/SIGDA 20th International Symposium on Field Programmable Gate Arrays*, pages 153–162, Monterey, CA, USA. 11, 29, 33, 34

[Auspy, 2014] Auspy (2014). Auspy Development Inc. Available from: `http://www.auspy.com`. 9, 20, 48

[Babb et al., 1997] Babb, J., Tessier, R., Dahl, M., Hanono, S., Hoki, D., and Agarwal, A. (1997). Logic Emulation with Virtual Wires. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(6):609–626. 34

[Bilgic, 1982] Bilgic, O. (1982). A synchronous frequency multiplier using phase-locked loop. *International Journal of Electronics*, 52(6):569–573. 36

[Bui and Moon, 1994] Bui, T. and Moon, B. (1994). A Fast and Stable Hybrid Genetic Algorithm for the Ratio-Cut Partitioning Problem on Hypergraphs. In *Proceedings of the 31st Conference on Design Automation*, pages 664–669, San Diego, CA, USA. 46

[Cadence, 2011] Cadence (2011). ASIC Prototyping Simplified. Technical report, Cadence Design Systems. 26, 32, 64

[Chan et al., 1994] Chan, P., Schlag, M., and Zien, J. (1994). Spectral K-Way Ratio-Cut Partitioning and Clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(9):1088–1096. 46

[Chou et al., 1994] Chou, N., Liu, L., Cheng, C., Dai, W., and Lindelof, R. (1994). Circuit Partitioning for Huge Logic Emulation Systems. In *Proceedings of the 31st Conference on Design Automation*, pages 244–249, San Diego, CA, USA. 47

[Cong et al., 1997] Cong, J., Li, H., Lim, S., Shibuya, T., and Xu, D. (1997). Large Scale Circuit Partitioning With Loose/Stable Net Removal And Signal Flow Based Clustering. In *Proceedings of International Conference on Computer-Aided Design*, pages 441–446, San Jose, CA, USA. 47

[Cong and Lim, 2000] Cong, J. and Lim, S. (2000). Edge Separability Based Circuit Clustering with Application to Circuit Partitioning. In *Proceedings of International Conference on Asia and South Pacific Design Automation*, pages 429–434, Yokohama, Japan. 47

[Cormen et al., 2009] Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2009). *Introduction to Algorithms*. The MIT Press. 21, 52

[DINI, 2014] DINI (2014). The dini group. Available from: `http://www.dinigroup.com/new/products.php`. 9, 12, 13, 18, 19, 23, 39, 40, 43, 51, 55, 65

[Drechsler et al., 2002] Drechsler, R., Gunther, W., Eschbach, T., Linhard, L., and Angst, G. (2002). Recursive Bi-Partitioning of Netlists for Large Number of Partitions. In *Proceedings of Euromicro Symposium on Digital Systems Design*, pages 38–44, Dortmund, Germany. 47

[Dutt, 1993] Dutt, S. (1993). New Faster Kernighan-Lin-Type Graph-Partitioning Algorithms. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 370–377, Santa Clara, CA, USA. 46

[Ejnioui and Ranganathan, 2003] Ejnioui, A. and Ranganathan, N. (2003). Multiterminal Net Routing for Partial Crossbar-Based Multi-FPGA Systems. *IEEE Transaction on very large scale integration (VLSI) systems*, 11(1):71–78. 22

[Fiduccia and Mattheyeses, 1982] Fiduccia, C. and Mattheyeses, R. (1982). A Linear-time Heuristic for Improving Network Partitions. In *Proceedings of the 19th Design Automation Conference*, pages 175–181, Las Vegas, Nevada, USA. 46, 47

[Flexras, 2014] Flexras (2014). Flexras Technologies. Available from: `http://www.flexras.com`. 9, 20, 48, 56, 68, 74, 79, 88, 89, 106, 107, 108

[FSP, 2014] FSP (2014). Allegro FPGA System Planner. Available from: `http://www.cadence.com/products/pcb/fpga_planner/pages/default.aspx`. 26, 61, 65, 89

[Gaisler, 2014] Gaisler (2014). Gaisler Research. Available from: `http://www.gaisler.com`. 43, 55, 62, 94, 98, 112, 118

[Garey and Johnson, 1990] Garey, M. and Johnson, D. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. Freeman. 47, 74

[GiDEL, 2014] GiDEL (2014). Available from: `http://www.gidel.com`. 12

[Hagen and Kahng, 1992] Hagen, L. and Kahng, A. (1992). New Spectral Methods for Ratio Cut Partitioning and Clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(9):1074–1085. 46

[HAPS, 2014] HAPS (2014). HAPS Family of FPGA-Based Prototyping Solutions. Available from: `http://www.synopsys.com/Systems/FPGABasedPrototyping/Pages/HAPS.aspx`. 9, 12, 30, 39, 104, 114

[Hauck, 1995] Hauck, S. (1995). *Multi-FPGA Systems*. PhD thesis, University of Washington, Seattle, WA, USA. Available from: `http://www.ee.washington.edu/faculty/hauck/Thesis/`. 28

[Hauck and Borriello, 1995] Hauck, S. and Borriello, G. (1995). Logic Partition Orderings for Multi-FPGA Systems. In *Proceedings of the 3rd International ACM Symposium on Field-Programmable Gate Arrays*, pages 32–38, Monterey, CA, USA. 47

[Huang et al., 2011] Huang, C., Yin, Y., and Hsu, C. (2011). SoC HW/SW Verification and Validation. In *Proceedings of the 16th Asia South Pacific Design Automation Conference*, pages 297–300, Yokohama, Japan. 10

[Huang and Kahng, 1995] Huang, D. and Kahng, A. (1995). Multi-Way System Partitioning into a Single Type or Multiple Types of FPGAs. In *Proceedings of the 3rd International ACM Symposium on Field-Programmable Gate Arrays*, pages 140–145, Monterey, CA, USA. 47

[Hyder and Wawrzynek, 2005] Hyder, Z. and Wawrzynek, J. (2005). Defect Tolerance in Multiple-FPGA Systems. In *Proceedings of the 2005 International Conference on Field Programmable Logic and Applications*, pages 247–254, Tampere, Finland. 33

[HyperSilicon, 2014] HyperSilicon (2014). Fpga prototype, soc verification. Available from: `http://www.hypersilicon.com/English/Product.asp`. 9, 19

[IBS, 2009] IBS (2009). International Business Strategies, Inc. Available from: `http://www.ibs-inc.net`. 7

[Inagi et al., 2010] Inagi, M., Takashima, Y., and Nakamura, Y. (2010). Globally Optimal Time-multiplexing of Inter-FPGA Connections for Multi-FPGA Prototyping Systems. *IPSJ Transactions on System LSI Design Methodology*, 3:81–90. 34, 40

[Incisive, 2014] Incisive (2014). Incisive Enterprise Simulator. Available from: `http://www.cadence.com/products/fv/enterprise_simulator/pages/default.aspx`. 8

[ISE, 2014] ISE (2014). ISE Design Suite. Available from: http://www.xilinx.com/support/index.html/content/xilinx/en/supportNav/design_tools/ise_design_suite.html. 45

[Jain et al., 2002] Jain, S., Kumar, A., and Kumar, S. (2002). Hybrid Multi-FPGA board evaluation by limiting multi-hop routing. In *Proceedings of the 13th IEEE International Workshop on Rapid System Prototyping*, pages 66–73, Darmstadt, Germany. 23

[Karypis et al., 1997] Karypis, G., Aggarwal, R., Kumar, V., and Shekhar, S. (1997). Multilevel Hypergraph Partitioning: Application in VLSI Domain. In *Proceedings of the 34st Conference on Design Automation*, pages 526–529, Anaheim, CA, USA. 47

[Karypis and Kumar, 1998] Karypis, G. and Kumar, V. (1998). hMetis 1.5: A Hypergraph Partitioning Package. Technical report, Department of Computer Science, University of Minnesota. 47

[Kernighan and Lin, 1970] Kernighan, W. and Lin, S. (1970). An Efficient Heuristic Procedure for Partitioning of Electrical Circuits. *Bell Systems Technical Journal*, 49(2):291–307. 46

[Khalid, 1999] Khalid, M. (1999). *Routing Architecture and Layout Synthesis for Multi-FPGA Systems*. PhD thesis, University of Toronto, Toronto, Canada. Available from: http://www.collectionscanada.gc.ca/obj/s4/f2/dsk1/tape8/PQDD_0004/NQ41187.pdf. 22

[Khalid and Rose, 2000] Khalid, M. and Rose, J. (2000). A Novel and Efficient Routing Architecture for Multi-FPGA Systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(1):30–39. 22

[Krishnamurthy, 1984] Krishnamurthy, B. (1984). An Improved Min-Cut Algorithm for Partitioning VLSI Networks. *IEEE Transactions on Computers*, C-33(5):438–446. 46

[Krupnova, 2004] Krupnova, H. (2004). Mapping Multi-Million Gate SoCs on FPGAs: Industrial Methodology and Experience. In *Proceedings of Design, Automation and Test in Europe Conference and Exposition*, pages 1236–1241, Paris, France. 11, 33, 34

[Kulmala et al., 2007] Kulmala, A., Salminen, E., and Hamalainen, T. (2007). Evaluating Large System-on-Chip on Multi-FPGA Platform. In *Proceedings of the 7th International Workshop Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 179–189, Samos, Greece. 29, 33

[Kuon and Rose, 2010] Kuon, I. and Rose, J. (2010). *Quantifying and Exploring the Gap Between FPGAs and ASICs*. Springer, New York. 11, 46

[Kuznar et al., 1993] Kuznar, R., Brglez, F., and Kozminski, K. (1993). Cost Minimization of Partitions into Multiple Devices. In *Proceedings of the 30th Design Automation Conference*, pages 315–320, Dallas, TX, USA. 47

[Kuznar et al., 1994] Kuznar, R., Brglez, F., and Zajc, B. (1994). Multi-way Netlist Partitioning into Heterogeneous FPGAs and Minimization of Total Device Cost and Interconnect. In *Proceedings of the 31st Conference on Design Automation*, pages 238–243, San Diego, CA, USA. 47

[LFSR, 2014] LFSR (2014). Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators. Available from: `http://www.xilinx.com/support/documentation/application_notes/xapp052.pdf`. 40, 51

[Mak and Wong, 1997] Mak, W. and Wong, D. (1997). Board-Level Multiterminal Net Routing for FPGA-Based Logic Emulation. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2(2):151–167. 22

[McMurchie and Ebeling, 1995] McMurchie, L. and Ebeling, C. (1995). PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs. In *Proceedings of the 3rd International ACM Symposium on Field-Programmable Gate Arrays*, pages 111–117. 21, 52

[MGT, 2014] MGT (2014). 7 Series FPGAs GTX/GTH Transceivers User Guide. Available from: `http://www.xilinx.com/support/documentation/user_guides/ug476_7Series_Transceivers.pdf`. 36

[ModelSim, 2014] ModelSim (2014). ModelSim/SE. Available from: `http://www.mentor.com/products/fv/modelsim/`. 8

[OpenCores, 2014] OpenCores (2014). Available from: `http://www.opencores.org`. 43, 56, 62, 94

[Package, 2014] Package (2014). 7 Series FPGAs Packaging and Pinout Product Specification. Available from: `http://www.xilinx.com/support/documentation/user_guides/ug475_7Series_Pkg_Pinout.pdf`. 105

[Palladium, 2014] Palladium (2014). Cadence Palladium series. Available from: `http://www.cadence.com/products/sd/palladium_series/pages/default.aspx`. 8

[Pinmux, 2014] Pinmux (2014). Pin Multiplexing White Paper and IP. Available from: `http://www.dinigroup.com/new/pinmux_xilinx.php`. 34, 36, 37, 38, 39

[POLARIS, 2014] POLARIS (2014). Available from: `http://www.polaris-ds.com`. 12

[Posner, 2013a] Posner, M. (2013a). Cables Vs. PCB Traces. Available from: `http://blogs.synopsys.com/breakingthethreelaws/2013/03/ufc-cables-vs-pcb-traces/`. 111

[Posner, 2013b] Posner, M. (2013b). Ddr3 memory and how useful a brick can be. Available from: `http://blogs.synopsys.com/breakingthethreelaws/2013/04/ddr3-memory-and-how-useful-a-brick-can-be/`. 114

[Precision, 2014] Precision (2014). Advanced FPGA Synthesis. Available from: http://www.mentor.com/products/fpga/synthesis/. 46, 68, 88, 106

[Prodesign, 2014] Prodesign (2014). FPGA Prototyping. Available from: http://www.prodesign-europe.com/profpga/. 12, 29, 39, 104

[Protium, 2014] Protium (2014). Protium Rapid Prototyping Platform. Available from: http://www.cadence.com/products/sd/protium_rapid_prototyping/pages/default.aspx. 9, 19, 20, 23, 48, 65

[ProtoCompiler, 2014] ProtoCompiler (2014). Synopsys ProtoCompiler. Available from: http://www.synopsys.com/Systems/FPGABasedPrototyping/Pages/protocompiler.aspx. 9, 20

[Quartus, 2014] Quartus (2014). Design Entry and Synthesis. Available from: http://www.altera.com/products/software/quartus-ii/subscription-edition/design-entry-synthesis/qts-des-ent-syn.html. 46, 68, 88, 106

[ReFLEX, 2014] ReFLEX (2014). Available from: http://reflexces.com. 12, 23

[Riess et al., 1994] Riess, B., Doll, K., and Johannes, F. (1994). Partitioning Very Large Circuits Using Analytical Placement Techniques. In *Proceedings of the 31st Conference on Design Automation*, pages 646–651, San Diego, CA, USA. 46

[Roy and Sechen, 1993] Roy, K. and Sechen, C. (1993). A Timing Driven N-Way Chip and Multi-Chip Partitioner. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers*, pages 240–247, Santa Clara, CA, USA. 47

[S2C, 2014] S2C (2014). Rapid fpga-based soc & asic prototyping. Available from: http://www.s2cinc.com. 9, 19, 20

[Sanchis, 1989] Sanchis, L. (1989). Multiple-Way Network Partitions. *IEEE Transactions on Computers*, 38(1):62–81. 46

[Schelle et al., 2010] Schelle, G., Collins, J., Schuchman, E., Wang, P., Zou, X., Chinya, G., Plate, R., Mattner, T., Olbrich, F., Hammarlund, P., Singhal, R., Brayton, J., Steibl, S., and Wang, H. (2010). Intel nehalem processor core made FPGA synthesizable. In *Proceedings of the ACM/SIGDA 18th International Symposium on Field Programmable Gate Arrays*, pages 3–12, Monterey, CA, USA. 34

[Schirrmeister, 2014] Schirrmeister, F. (2014). DeepChip. Available from: http://www.deepchip.com/items/0517-06.html. 12

[Sekhar, 2014] Sekhar, V. (2014). Prototyping using FPGAs in VLSI design. Available from: http://www.unistring.com/images/IETE_WGL/FPGA-Prototyping-presentation%20.pdf. 13, 61, 65

[SelectIO, 2014] SelectIO (2014). 7 Series FPGAs SelectIO Resources User Guide. Available from: http://www.xilinx.com/support/documentation/user_guides/ug471_7Series_SelectIO.pdf. 37, 38, 39, 64, 105

[Selvakkumaran et al., 2004] Selvakkumaran, N., Ranjan, A., Raje, S., and Karypis, G. (2004). Multi-Resource Aware Partitioning Algorithms for FPGAs with Heterogeneous Resources. In *Proceedings of the 41th International Conference on Design Automation*, pages 741–746, San Diego, CA, USA. 46, 48

[SerialIO, 2014] SerialIO (2014). High-Speed Serial I/O Made Simple. Available from: http://www.xilinx.com/publications/archives/books/serialio.pdf. 36, 37

[Song et al., 2003] Song, X., Hung, W., Mishchenko, A., Chrzanowska-Jeske, M., Kennings, A., and Coppola, A. (2003). Board-Level Multiterminal Net Assignment for the Partial Cross-Bar Architecture. *IEEE Transaction on very large scale integration (VLSI) systems*, 11(3):511–514. 22

[Synopsys, 2014] Synopsys (2014). Certify. Available from: http://www.synopsys.com/Systems/FPGABasedPrototyping/Pages/Certify.aspx. 48

[Synplify, 2014] Synplify (2014). Logic Synthesis for FPGA Implementation. Available from: http://www.xilinx.com/tools/xst.htm. 46, 68, 88, 106

[Tang et al., 2014] Tang, Q., Tuna, M., and Mehrez, H. (2014). Future Inter-FPGA Communication Architecture for Multi-FPGA Based Prototyping. In *Proceedings of the ACM/SIGDA 22nd International Symposium on Field Programmable Gate Arrays*, page 251, Monterey, CA, USA. 36, 127

[Turki et al., 2013] Turki, M., Marrakchi, Z., Mehrez, H., and Abid, M. (2013). Iterative Routing Algorithm of Inter-FPGA Signals for Multi-FPGA Prototyping Platform. In *Proceedings of the 9th International Symposium on Applied Reconfigurable Computing (ARC)*, pages 210–217, Los Angeles, CA, USA. 20, 48, 50, 57, 89, 108

[Varghese et al., 1993] Varghese, J., Butts, M., and Batcheller, J. (1993). An efficient logic emulation system. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 1(2):171–174. 22

[VCS, 2014] VCS (2014). Functional Verification Solution. Available from: http://www.synopsys.com/Tools/Verification/FunctionalVerification/Pages/VCS.aspx. 8

[Veloce, 2014] Veloce (2014). Veloce Emulation Systems. Available from: http://www.mentor.com/products/fv/emulation-systems/. 8

[Vijayan, 1990] Vijayan, G. (1990). Partitioning Logic on Graph Structures to Minimize Routing Cost. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(12):1326–1334. 47

[Virtex-4, 2014] Virtex-4 (2014). Virtex-4 Family Overview. Available from: http://www.xilinx.com/support/documentation/data_sheets/ds112.pdf. 126

[Virtex-5, 2014] Virtex-5 (2014). Virtex-5 Family Overview. Available from: http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf. 55, 126

[Virtex-6, 2014] Virtex-6 (2014). Virtex-6 Family Overview. Available from: http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf. 126

[Virtex-7, 2014] Virtex-7 (2014). 7 Series FPGAs Overview. Available from: http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf. 126

[Virtual, 2014] Virtual (2014). Cadence Virtual System Platform. Available from: http://www.cadence.com/products/sd/virtual_system/pages/default.aspx. 9

[Virtualizer, 2014] Virtualizer (2014). Synopsys Virtualizer. Available from: http://www.synopsys.com/Systems/VirtualPrototyping/Pages/Virtualizer.aspx. 9

[Vista, 2014] Vista (2014). Electronic System Level Design. Available from: http://www.mentor.com/esl/. 9

[Walters, 1991] Walters, S. (1991). Computer-aided prototyping for ASIC-based systems. *IEEE Design and Test of Computers*, 8(2):4–10. 20

[Wichlund and Aas, 1998] Wichlund, S. and Aas, E. (1998). On Multilevel Circuit Partitioning. In *Proceedings of International Conference on Computer-Aided Design*, pages 505–511, San Jose, CA, USA. 47

[Woo and Kim, 1993] Woo, N. and Kim, J. (1993). An Efficient Method of Partitioning Circuits for Multiple-FPGA Implementation. In *Proceedings of the 30th Design Automation Conference*, pages 202–207, Dallas, TX, USA. 46

[Xilinx, 2014] Xilinx (2014). All programmable technologies from xilinx inc. Available from: http://www.xilinx.com. 13, 25, 31, 61, 67, 75, 81, 93

[XST, 2014] XST (2014). XST Synthesis. Available from: http://www.xilinx.com/tools/xst.htm. 46, 68, 88, 106

[Yang and Wong, 1994] Yang, H. and Wong, D. (1994). Efficient Network Flow Based Min-Cut Balanced Partitioning. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 50–55, San Jose, CA, USA. 46

[Yarack and Carletta, 2000] Yarack, E. and Carletta, J. (2000). An Evaluation of Move-Based Multi-Way Partitioning Algorithms. In *Proceedings of the IEEE International Conference on Computer Design*, pages 363–369, Austin, TX, USA. 46

[ZeBu, 2014] ZeBu (2014). ZeBu Emulation. Available from: http://www.synopsys.com/Tools/Verification/hardware-verification/emulation/Pages/default.aspx. 8